Broadview
www.broadview.com.cn

疯狂软件教育 www.fkjava.org 标准教材

本书第一版长期雄踞各网店Android图书销量排行榜前列

疯狂Android讲义（第2版）

李刚 编著

疯狂源自梦想

技术成就辉煌

疯狂源自梦想

技术成就辉煌

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

疯狂源自梦想 技术成就辉煌



看得懂 学得会 做得出

1. 知识全面，覆盖面广

本书深入阐述了Android应用开发的Activity、Service、BroadcastReceiver与ContentProvider四大组件，并详细介绍了Android全部图形界面组件的功能和用法，Android各种资源的管理与用法，Android图形、图像处理，事件处理，Android输入/输出处理，音频/视频等多媒体开发，OpenGL-ES开发，网络通信，传感器和GPS开发等内容，全面覆盖Android官方指南，在某些内容上更加具体、深入。

2. 内容实际，实用性强

本书并不局限于枯燥的理论介绍，而是采用了“项目驱动”的方式来讲授知识点，全书近百个实例，几乎每个知识点都可找到对应的参考实例。本书最后还提供了“疯狂连连看”、“电子拍卖系统Android客户端”两个应用，具有极高的参考价值。

3. 讲解详细，上手容易

本书保持了“疯狂Java体系”的一贯风格：操作步骤详细、编程思路清晰，语言平实。只要读者有一定的Java编程基础，阅读本书将可以很轻松地上手Android应用开发；学习完本书最后的两个案例后，读者即可完全满足实际企业中Android应用开发的要求。

阅读此书有任何技术问题，都可以登录如下站点获得解决：
疯狂Java联盟：http://www.crazyit.org

上架建议：移动开发>Android

ISBN 978-7-121-19485-6

9 787121 194856 >

定价：99.00元（含光盘1张）

策划编辑：张月萍
责任编辑：高洪霞
封面设计：李 玲

疯狂 Android讲义（第2版）

李刚 编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

移动互联网已经成为当今世界发展最快、市场潜力最大、前景最诱人的业务，而 Android 则是移动互联网上市场占有率最高的平台（已远超 iOS，最新统计数据：Android 占 53.7%，iOS 占 35%）；与此同时，Android 应用选择了 Java 作为其开发语言，这对于 Java 来说也是一次极好的机会。

本书是《疯狂 Android 讲义》的第 2 版。本书基于最新的 Android 4.2，Android SDK、ADT 都基于 Android 4.2，书中每个案例、每个截图都全面升级到 Android 4.2。本书全面地介绍了 Android 应用开发的相关知识，全书内容覆盖了 Android 用户界面编程、Android 四大组件、Android 资源访问、图形/图像处理、事件处理机制、Android 输入/输出处理、音频/视频多媒体应用开发、OpenGL 与 3D 应用开发、网络通信编程、Android 平台的 Web Service、传感器应用开发、GPS 应用开发、Google Map 服务等。

本书并不局限于介绍 Android 编程的各种理论知识，而是从“项目驱动”的角度来讲授理论。全书一共包括近百个实例，这些示范性的实例既可帮读者更好地理解各知识点在实际开发中的应用，也可供读者在实际开发时作为参考、拿来就用。本书最后还提供了两个实用的案例：疯狂连连看和电子拍卖系统 Android 客户端，具有极高的参考价值。本书提供了配套的答疑网站，如果读者在阅读本书时遇到技术问题，可以登录疯狂 Java 联盟（http://www.crazyit.org）发帖，笔者将会及时予以解答。

本书适合有一定 Java 编程基础的读者。如果读者已熟练掌握 Java 编程语法并具有一定图形界面编程经验，阅读本书将十分合适。否则，阅读本书之前建议先认真阅读疯狂 Java 体系之《疯狂 Java 讲义》。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

疯狂 Android 讲义 / 李刚编著. —2 版. —北京：电子工业出版社，2013.3
ISBN 978-7-121-19485-6

Ⅰ. ①疯… Ⅱ. ①李… Ⅲ. ①移动电话机－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2013）第 017791 号

策划编辑：张月萍
责任编辑：高洪霞
印　　刷：北京中新伟业印刷有限公司
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
开　　本：787×1092　1/16　印张：49.75　字数：1273 千字　彩插：1
印　　次：2013 年 4 月第 2 次印刷
印　　数：5001～10000 册　定价：99.00 元（含光盘 1 张）

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。



前　言

移动互联网热潮在全世界引起了巨大反响，移动互联网正在改变着传统互联网的格局，全世界的IT公司争相将业务重心向移动互联网转移，移动互联网业务也成为业内最大的利润增长点。

Android系统就是一个开放式的移动互联网操作系统，Android已经成为应用最广的移动互联网平台（远超Apple公司的iOS和Microsoft的Windows Phone，最新统计数据：Android占53.7%，iOS占35%）。

对于Java语言而言，Android系统给了Java一个新的机会。在过去的岁月中，Java语言作为服务器端编程语言，已经取得了极大的成功，Java EE平台发展得非常成熟，而且一直是电信、移动、银行、证券、电子商务应用的首选平台、不争的王者。但在客户端应用开发方面，Java语言一直表现不佳，虽然Java既有AWT/Swing界面开发库，也有SWT/JFace界面开发库，但对于客户端应用开发人员而言，大多不愿意选择Java语言。Android系统的出现改变了这种局面，Android是一个非常优秀的手机、平板电脑操作系统，正不断蚕食传统的桌面操作系统，而Android平台应用的开发语言就是Java，这意味着Java语言将可以在客户端应用开发上大展拳脚。

Android已经成为应用最广的手机、平板电脑操作系统，采用Java语言开发的Android应用也越来越多。不过需要指出的是，运行Android平台的硬件只是手机、平板电脑等便携式设备，这些设备的计算能力、数据存储能力都是有限的，因此不太可能在Android平台上部署大型企业级应用，因此Android应用可能以纯粹客户端应用的角色出现，然后通过网络与传统大型应用交互，充当大型企业应用的客户端，比如现在已经出现的淘宝Android客户端、赶集网Android客户端，它们都是这种发展趋势下的产物。

对于Java开发者来说，以前主要在Java EE平台上从事服务器端应用开发，但在移动互联网的趋势下，Java开发者必然面临着为这些应用开发客户端的需求。对于Java开发者来说，Android应用开发既是一个挑战，也是一个机遇——挑战是：掌握Android应用开发需要重新投入学习成本；机遇是：掌握Android开发之后将可让职业生涯达到一个新的高度，而且移动互联网与Android必然带来更多的就业机会与创业机会，这都值得当下的开发者好好把握。

本书是《疯狂Android讲义》的第2版，本书真正基于最新的Android 4.2，Android SDK、ADT都基于Android 4.2。书中每个案例、每个截图都是基于Android 4.2，全面介绍了Switch、ActionBar、Fragment、FragmentActivity、属性动画等为平板电脑新增的特性。

衷心感谢



疯狂Java体系图书能走到今天，广大读者的认同与支持是笔者坚持创作的最大动力。广大读者的认同已让疯狂Java体系图书的销量稳居国内第一。《疯狂Android讲义》于2011年

7 月上市，该书创造了“首印 20 天售罄、每个月重印一次”的奇迹，这些必须感谢广大读者的认同与支持。

《疯狂 Android 讲义》在所有 Android 图书销量稳居榜首，在京东网搜索“android”关键字相关的图书，可以看到《疯狂 Android 讲义》实际排在第 1 位（第 1 位并非介绍 Android 开发的图书），截图如下。



在亚马逊网搜索“android”关键字相关的图书，可以看到《疯狂 Android 讲义》排在第 1 位，截图如下。



诚挚地感谢广大读者的支持与爱护：你们的支持让疯狂 Java 图书没有放弃，你们的激励让疯狂 Java 图书茁壮成长，你们的反馈让疯狂 Java 图书日臻完善；同时也感谢博文视点张月萍等编辑、疯狂软件教育中心技术团队一贯的支持。

本书有什么特点



本书是一本介绍 Android 应用开发的实用图书，全面介绍了 Android 4.2 平台上应用开发各方面的知识。与市面上有些介绍 Android 编程的图书不同，本书并没有花太多篇幅介绍 Android 的发展历史（因为这些内容到处都是），完全没有介绍 Android 市场（因为它只是一个交易网站，与 Android 开发无关，但有些图书甚至用整整一章来介绍它），也没有介绍 JDK 安装、环境变量配置等内容——笔者假设读者已经具有一定的 Java 功底。换句话来说，如果你对 JDK 安装、Java 基本语法还不熟，本书并不适合你。

本书只用了一章来介绍如何搭建 Android 开发环境、Android 应用结构，当然也简要说明了 Android 的发展历史。可能依然会有人觉得本书篇幅很多，这是由于本书覆盖了 Android 开发绝大部分知识，而且很多知识不仅介绍了相应的理论，并通过相应的实例程序给出了示范。

需要说明的是，本书只是一本介绍 Android 实际开发的图书，这不是一本关于所谓“思想”的书，不要指望学习本书能提高你所谓的“Android 思想”，所以奉劝那些希望提高编程思想的读者不要阅读本书。

本书更不是一本看完之后可以“吹嘘、炫耀”的书——因为本书并没有堆砌一堆“深奥”的新名词、一堆“高深”的思想，本书保持了“疯狂 Java 体系”的一贯风格：操作步骤详细，编程思路清晰，语言平实。只要读者有基本的 Java 基础，阅读本书不会有任何问题，看完本书不会让你觉得自己突然“高深”了，“高深”到自己都理解不了。

认真看完本书、把书中所有示例都练习一遍，本书带给你的只是 9 个字：“看得懂、学得会、做得出”。本书不能让你认识一堆新名词，只会让你学会实际的 Android 应用开发。

如果读者有非常扎实的 Java 基本功、良好的英文阅读能力，而且对图形用户界面编程也有丰富的经验，不管是 AWT/Swing 编程的经验，还是 SWT 编程的经验，抑或是 Windows 界面编程的经验都行，那没有多大必要购买本书，只要花几天时间快速浏览本书即可动手编程了。如果遇到某个类、某个功能不太确定，直接查阅 Android Dev Guide 和 API 参考文档即可。

不管怎样，只要读者在阅读本书时遇到知识上的问题，都可以登录疯狂 Java 联盟（http://www.crazyit.org）与广大 Java 学习者交流，笔者也会通过该平台与大家一起交流、学习。

本书还具有如下几个特点。

1. 知识全面，覆盖面广

本书深入阐述了 Android 应用开发的 Activity、Service、BroadcastReceiver 与 ContentProvider 四大组件，并详细介绍了 Android 全部图形界面组件的功能和用法，Android 各种资源的管理与用法，Android 图形、图像处理，事件处理，Android 输入/输出处理，音频

/视频等多媒体开发，OpenGL-ES开发，网络通信，传感器和GPS开发等内容，全面覆盖Android官方指南，在某些内容上更加具体、深入。

2．内容实际，实用性强

本书并不局限于枯燥的理论介绍，而是采用了“项目驱动”的方式来讲授知识点，全书近百个实例，几乎每个知识点都可找到对应的参考实例。本书最后还提供了“疯狂连连看”、“电子拍卖系统Android客户端”两个应用，具有极高的参考价值。

3．讲解详细，上手容易

本书保持了“疯狂Java体系”的一贯风格：操作步骤详细，编程思路清晰，语言平实。只要读者有一定的Java编程基础，阅读本书将可以很轻松地上手Android应用开发；学习完本书最后的两个案例后，读者即可完全满足实际企业中Android应用开发的要求。

本书写给谁看



如果你已经具备一定的Java基础和XML基础，或已经学完了《疯狂Java讲义》一书，那么你阅读此书将会比较适合；如果你有不错的Java基础，而且有一定的图形界面编程经验，那么阅读本书将可以很快掌握Android应用开发；如果你对Java的掌握还不熟练，比如对JDK安装、Java基本语法都不熟练，建议遵从学习规律，循序渐进，暂时不要购买、阅读此书。

光盘中有哪些内容



1．光盘内容

（1）书中的代码在光盘中按章、节存放，01～19个文件夹名对应于本书中的章名，即第2章所使用的代码放在codes文件夹的02文件夹下，依此类推。书中源代码也给出光盘路径，方便读者查找。

（2）本书的绝大部分项目都是Eclipse项目，因此项目文件夹下包含.classpath、.project等文件，它们是Eclipse项目文件，请不要删除。

2．运行环境

（1）安装jdk-7u5-windows-i586-p.exe，安装完成后，添加CLASSPATH环境变量，该环境变量的值为.;%JAVA_HOME%/lib/tools.jar;%JAVA_HOME%/lib/dt.jar。为了可以编译和运行Java程序，还应该在PATH环境变量中增加%JAVA_HOME%/bin。其中JAVA_HOME代表JDK（不是JRE）的安装路径。

（2）安装Android 4.2。创建AVD虚拟设备。安装Android SDK的方法请参考本书第1章。

（3）安装Apache的Tomcat7.0.30，不要使用安装文件安装，而是采用解压缩的安装方式。安装Tomcat请参看疯狂Java体系的《轻量级Java EE企业应用实战》第1章。

安装完成后，将Tomcat安装路径的lib下的jsp-api.jar和servlet-api.jar两个JAR文件添加到CLASSPATH环境变量之后。

（4）安装 apache-ant-1.8.2。将下载的 Ant 压缩文件解压缩到任意路径，然后增加 ANT_HOME 的环境变量，让变量的值为 Ant 的解压缩路径。并在 PATH 环境变量中增加 %ANT_HOME%/bin 环境变量。

（5）安装 Eclipse-jee-juno 版（也就是 Eclipse 4.2 for Java EE Developers），并安装 ADT 插件，安装插件后在 Eclipse 中设置 Android SDK 的路径。

3. 注意事项

（1）本书所有 Android 项目都是 Eclipse 工程，读者只要将它们导入 Eclipse 工具中即可。

（2）介绍网络编程章节涉及少数 Web 应用，将该 Web 应用复制到%TOMCAT_HOME%/webapps 路径下，然后进入 build.xml 所在路径，执行如下命令：

```
ant compile -- 编译应用
```

启动 Tomcat 服务器，使用浏览器即可访问该应用。

（3）对于 Eclipse 项目文件，导入 Eclipse 开发工具即可。

（4）第 19 章的案例，请参看项目下的 readme.txt。

（5）本书有部分案例需要连接数据库，读者应修改数据库 URL 及用户名、密码让这些代码与读者运行环境一致。如果项目下有 SQL 脚本，导入 SQL 脚本即可，如果没有 SQL 脚本，系统将在运行时自动建表，读者只需创建对应数据库即可。

（6）本书关于网络编程、传感器编程等部分章节需要连接 PC。笔者 PC 的 IP 地址为 192.168.1.88，读者可以将自己的 IP 地址设为该地址，或将程序中用到该 IP 地址的地方修改为自己的 PC 的 IP 地址。

（7）在使用本光盘的程序时，请将程序复制到硬盘上，并去除文件的只读属性。

4. 技术支持

使用本光盘时若遇到技术问题，可登录 http://www.crazyit.org 与作者联系。



2013 年 1 月

目 录 CONTENTS

第 1 章 Android 应用与开发环境 ····· 1

1.1 Android 的发展和历史 ····· 2

1.1.1 Android 的发展和简介 ····· 2

1.1.2 Android 平台架构及特性 ····· 3

1.2 搭建 Android 开发环境 ····· 5

1.2.1 下载和安装 Android SDK ····· 5

1.2.2 安装运行、调试环境 ····· 7

1.2.3 安装 Eclipse 和 ADT 插件 ····· 10

1.3 Android 常用开发工具的用法 ····· 13

1.3.1 在命令行创建、删除和浏览 AVD ····· 13

1.3.2 使用 Android 模拟器（Emulator） ····· 14

1.3.3 使用 DDMS 进行调试 ····· 15

1.3.4 Android Debug Bridge（ADB）的用法 ····· 16

1.3.5 使用 DX 编译 Android 应用 ····· 18

1.3.6 使用 Android Asset Packaging Tool（AAPT）打包资源 ····· 18

1.3.7 使用 mksdcard 管理虚拟 SD 卡 ··· 18

1.4 开始第一个 Android 应用 ····· 19

1.4.1 使用 Eclipse 开发第一个 Android 应用 ····· 19

1.4.2 通过 ADT 运行 Android 应用 ····· 23

1.5 Android 应用结构分析 ····· 23

1.5.1 创建一个 Android 应用 ····· 24

1.5.2 自动生成的 R.java ····· 26

1.5.3 res 目录说明 ····· 27

1.5.4 Android 应用的清单文件：AndroidManifest.xml ····· 28

1.5.5 应用程序权限说明 ····· 29

1.6 Android 应用的基本组件介绍 ····· 30

1.6.1 Activity 和 View ····· 30

1.6.2 Service ····· 31

1.6.3 BroadcastReceiver ····· 31

1.6.4 ContentProvider ····· 32

1.6.5 Intent 和 IntentFilter ····· 32

1.7 签名 Android 应用程序 ····· 33

1.7.1 在 Eclipse 中对 Android 应用签名 ····· 34

1.7.2 使用命令对 APK 包签名 ····· 35

1.8 本章小结 ····· 37

第 2 章 Android 应用的界面编程 ····· 38

2.1 界面编程与视图（View）组件 ····· 39

2.1.1 视图组件与容器组件 ····· 39

2.1.2 使用 XML 布局文件控制 UI 界面 ····· 44

2.1.3 在代码中控制 UI 界面 ····· 45

2.1.4 使用 XML 布局文件和 Java 代码混合控制 UI 界面 ····· 46

2.1.5 开发自定义 View ····· 47

2.2 第 1 组 UI 组件：布局管理器 ····· 50

2.2.1 线性布局 ····· 50

2.2.2 表格布局 ····· 53

2.2.3 帧布局 ····· 56

2.2.4 相对布局 ····· 59

2.2.5 Android 4.0 新增的网格布局 ····· 61

2.2.6 绝对布局 ····· 63

2.3 第 2 组 UI 组件：TextView 及其子类 ····· 65

2.3.1 文本框（TextView）与编辑框（EditText）的功能和用法 ····· 65

2.3.2 EditText 的功能与用法 ····· 72

2.3.3 按钮（Button）组件的功能和用法 ····· 74

2.3.4 使用 9Patch 图片作为按钮背景 ····· 76

2.3.5 单选按钮（RadioButton）与复选框（CheckBox）的功能与用法 ····· 77

2.3.6 状态开关按钮（ToggleButton）与开关（Switch）的功能与用法 ····· 79

2.3.7 时钟（AnalogClock 和 DigitalClock）的功能与用法 ····· 81

2.3.8 计时器（Chronometer）……83
2.4 第3组UI组件：ImageView及其子类……84
2.5 第4组UI组件：AdapterView及子类……91
2.5.1 列表视图（ListView）和ListActivity……91
2.5.2 Adapter接口及实现类……93
2.5.3 自动完成文本框（AutoCompleteTextView）的功能和用法……102
2.5.4 网格视图（GridView）功能和用法……104
2.5.5 可展开的列表组件（ExpandableListView）……107
2.5.6 Spinner的功能和用法……110
2.5.7 画廊视图（Gallery）的功能和用法……112
2.5.8 AdapterViewFlipper的功能与用法……114
2.5.9 StackView的功能与用法……117
2.6 第5组UI组件：ProgressBar及其子类……119
2.6.1 进度条（ProgressBar）的功能与用法……119
2.6.2 拖动条（SeekBar）的功能和用法……123
2.6.3 星级评分条（RatingBar）的功能和用法……125
2.7 第6组UI组件：ViewAnimator及其子类……126
2.7.1 ViewSwitcher的功能与用法……127
2.7.2 图像切换器（ImageSwitcher）的功能与用法……132
2.7.3 文本切换器（TextSwitcher）的功能与用法……134
2.7.4 ViewFlipper的功能与用法……136
2.8 各种杂项组件……138
2.8.1 使用Toast显示提示信息框……138
2.8.2 日历视图（CalendarView）组件的功能和用法……140
2.8.3 日期、时间选择器（DatePicker和TimePicker）的功能和用法……141
2.8.4 数值选择器（NumberPicker）的功能与用法……144
2.8.5 搜索框（SearchView）的功能与用法……146
2.8.6 选项卡（TabHost）的功能和用法……148
2.8.7 滚动视图（ScrollView）的功能和用法……150
2.8.8 Notification的功能与用法……151
2.9 对话框……154
2.9.1 使用AlertDialog创建对话框……154
2.9.2 对话框风格的窗口……161
2.9.3 使用PopupWindow……161
2.9.4 使用DatePickerDialog、TimePickerDialog……163
2.9.5 使用ProgressDialog创建进度对话框……164
2.10 菜单……167
2.10.1 选项菜单和子菜单（SubMenu）……167
2.10.2 使用监听器来监听菜单事件……171
2.10.3 创建复选菜单项和单选菜单项……171
2.10.4 设置与菜单项关联的Activity……171
2.10.5 上下文菜单……172
2.10.6 使用XML文件定义菜单……174
2.10.7 使用PopupMenu创建弹出式菜单……178
2.11 使用活动条（ActionBar）……179
2.11.1 启用ActionBar……180
2.11.2 使用ActionBar显示选项菜单……181
2.11.3 启用程序图标导航……182
2.11.4 添加Action View……184
2.11.5 使用ActionBar实现Tab导航……185
2.11.6 使用ActionBar实现下拉式导航……191

2.12 本章小结……192
第 3 章 Android 的事件处理……193
3.1 Android 事件处理概述……194
3.2 基于监听的事件处理……194
3.2.1 监听的处理模型……195
3.2.2 事件和事件监听器……197
3.2.3 内部类作为事件监听器类……200
3.2.4 外部类作为事件监听器类……200
3.2.5 Activity 本身作为事件监听器……202
3.2.6 匿名内部类作为事件监听器类……203
3.2.7 直接绑定到标签……204
3.3 基于回调的事件处理……205
3.3.1 回调机制与监听机制……205
3.3.2 基于回调的事件传播……206
3.3.3 重写 onTouchEvent 方法响应触摸屏事件……208
3.4 响应的系统设置的事件……210
3.4.1 Configuration 类简介……210
3.4.2 重写 onConfigurationChanged 响应系统设置更改……212
3.5 Handler 消息传递机制……214
3.5.1 Handler 类简介……214
3.5.2 Handler、Loop、MessageQueue 的工作原理……216
3.6 异步任务（AsyncTask）……220
3.7 本章小结……223
第 4 章 深入理解 Activity 与 Fragment……224
4.1 建立、配置和使用 Activity……225
4.1.1 Activity……225
4.1.2 配置 Activity……233
4.1.3 启动、关闭 Activity……235
4.1.4 使用 Bundle 在 Activity 之间交换数据……237
4.1.5 启动其他 Activity 并返回结果……241
4.2 Activity 的回调机制……245
4.3 Activity 的生命周期与加载模式……246
4.3.1 Activity 的生命周期演示……246
4.3.2 Activity 与 Servlet 的相似性与区别……250
4.3.3 Activity 的 4 种加载模式……251
4.4 Fragment 详解……257
4.4.1 Fragment 概述及其设计哲学……257
4.4.2 创建 Fragment……258
4.4.3 Fragment 与 Activity 通信……262
4.4.4 Fragment 管理与 Fragment 事务……264
4.5 Fragment 的生命周期……268
4.6 本章小结……272
第 5 章 使用 Intent 和 IntentFilter 进行通信……273
5.1 Intent 对象详解……274
5.1.1 使用 Intent 启动系统组件……274
5.2 Intent 的属性及 intent-filter 配置……275
5.2.1 Component 属性……275
5.2.2 Action、Category 属性与 intent-filter 配置……277
5.2.3 指定 Action、Category 调用系统 Activity……282
5.2.4 Data、Type 属性与 intent-filter 配置……287
5.2.5 Extra 属性……295
5.2.6 Flag 属性……295
5.3 使用 Intent 创建 Tab 页面……296
5.4 本章小结……297
第 6 章 Android 应用的资源……298
6.1 资源的类型及存储方式……299
6.1.1 资源的类型以及存储方式……299
6.1.2 使用资源……301
6.2 使用字符串、颜色、尺寸资源……302
6.2.1 颜色值的定义……303
6.2.2 定义字符串、颜色、尺寸资源文件……303
6.2.3 使用字符串、颜色、尺寸资源……305
6.3 数组（Array）资源……307
6.4 使用（Drawable）资源……310
6.4.1 图片资源……310

6.4.2 StateListDrawable 资源 ····· 311
6.4.3 LayerDrawable 资源 ····· 312
6.4.4 ShapeDrawable 资源 ····· 314
6.4.5 ClipDrawable 资源 ····· 316
6.4.6 AnimationDrawable 资源 ····· 318
6.5 属性动画（Property Animation）资源 ····· 320
6.6 使用原始 XML 资源 ····· 322
6.6.1 定义原始 XML 资源 ····· 322
6.6.2 使用原始 XML 文件 ····· 323
6.7 使用布局（Layout）资源 ····· 325
6.8 使用菜单（Menu）资源 ····· 325
6.9 样式（Style）和主题（Theme）资源 ····· 326
6.9.1 样式资源 ····· 326
6.9.2 主题资源 ····· 327
6.10 属性（Attribute）资源 ····· 329
6.11 使用原始资源 ····· 332
6.12 国际化和资源自适应 ····· 333
6.12.1 Java 国际化的思路 ····· 334
6.12.2 Java 支持的语言和国家 ····· 334
6.12.3 完成程序国际化 ····· 335
6.12.4 为 Android 应用提供国际化资源 ····· 337
6.12.5 国际化 Android 应用 ····· 338
6.13 自适应不同屏幕的资源 ····· 340
6.14 本章小结 ····· 343
第 7 章 图形与图像处理 ····· 344
7.1 使用简单图片 ····· 345
7.1.1 使用 Drawable 对象 ····· 345
7.1.2 Bitmap 和 BitmapFactory ····· 345
7.2 绘图 ····· 348
7.2.1 Android 绘图基础：Canvas、Paint 等 ····· 348
7.2.2 Path 类 ····· 352
7.2.3 绘制游戏动画 ····· 355
7.3 图形特效处理 ····· 362
7.3.1 使用 Matrix 控制变换 ····· 362
7.3.2 使用 drawBitmapMesh 扭曲图像 ····· 366
7.3.3 使用 Shader 填充图形 ····· 368
7.4 逐帧（Frame）动画 ····· 370
7.4.1 AnimationDrawable 与逐帧动画 ····· 371
7.5 补间（Tween）动画 ····· 374
7.5.1 Tween 动画与 Interpolator ····· 374
7.5.2 位置、大小、旋转度、透明度改变的补间动画 ····· 376
7.5.3 自定义补间动画 ····· 380
7.6 属性动画 ····· 383
7.6.1 属性动画的 API ····· 383
7.6.2 使用属性动画 ····· 385
7.7 使用 SurfaceView 实现动画 ····· 393
7.7.1 SurfaceView 的绘图机制 ····· 394
7.8 本章小结 ····· 398
第 8 章 Android 数据存储与 IO ····· 399
8.1 使用 SharedPreferences ····· 400
8.1.1 SharedPreferences 与 Editor 简介 ····· 400
8.1.2 SharedPreferences 的存储位置和格式 ····· 401
8.1.3 读、写其他应用 SharedPreferences ····· 403
8.2 File 存储 ····· 404
8.2.1 openFileOutput 和 openFileInput ····· 405
8.2.2 读写 SD 卡上的文件 ····· 407
8.3 SQLite 数据库 ····· 414
8.3.1 SQLiteDatabase 简介 ····· 414
8.3.2 创建数据库和表 ····· 415
8.3.3 使用 SQL 语句操作 SQLite 数据库 ····· 416
8.3.4 使用 sqlite3 工具 ····· 418
8.3.5 使用特定方法操作 SQLite 数据库 ····· 419
8.3.6 事务 ····· 422
8.3.7 SQLiteOpenHelper 类 ····· 422
8.4 手势（Gesture） ····· 427
8.4.1 手势检测 ····· 427
8.4.2 增加手势 ····· 434

8.4.3 识别用户的手势……437
8.5 自动朗读（TTS）……439
8.6 本章小结……441
第 9 章 使用 ContentProvider 实现数据共享……442
9.1 数据共享标准：ContentProvider 简介……443
9.1.1 ContentProvider 简介……443
9.1.2 Uri 简介……444
9.1.3 使用 ContentResolver 操作数据……445
9.2 开发 ContentProvider……446
9.2.1 ContentProvider 与 ContentResolver 的关系……446
9.2.2 开发 ContentProvider……447
9.2.3 配置 ContentProvider……448
9.2.4 使用 ContentResolver 调用方法……449
9.2.5 创建 ContentProvider 的说明……451
9.3 操作系统的 ContentProvider……457
9.3.1 使用 ContentProvider 管理联系人……457
9.3.2 使用 ContentProvider 管理多媒体内容……463
9.4 监听 ContentProvider 的数据改变……466
9.4.1 ContentObserver 简介……466
9.5 本章小结……468
第 10 章 Service 与 BroadcastReceiver……469
10.1 Service 简介……470
10.1.1 创建、配置 Service……470
10.1.2 启动和停止 Service……472
10.1.3 绑定本地 Service 并与之通信……473
10.1.4 Service 的生命周期……477
10.1.5 使用 IntentService……478
10.2 跨进程调用 Service（AIDL Service）……481
10.2.1 AIDL Service 简介……482
10.2.2 创建 AIDL 文件……482
10.2.3 将接口暴露给客户端……483
10.2.4 客户端访问 AIDLService……484
10.3 电话管理器（TelephonyManager）……491
10.4 短信管理器（SmsManager）……498
10.5 音频管理器（AudioManager）……502
10.5.1 AudioManager 简介……502
10.6 振动器（Vibrator）……504
10.6.1 Vibrator 简介……504
10.6.2 使用 Vibrator 控制手机振动……505
10.7 手机闹钟服务（AlarmManager）……505
10.7.1 AlarmManager 简介……505
10.7.2 设置闹钟……506
10.8 接收广播消息……510
10.8.1 BroadcastReceiver 简介……510
10.8.2 发送广播……512
10.8.3 有序广播……513
10.9 接收系统广播消息……520
10.10 本章小结……523
第 11 章 多媒体应用开发……524
11.1 音频和视频的播放……525
11.1.1 使用 MediaPlayer 播放音频……525
11.1.2 音乐特效控制……528
11.1.3 使用 SoundPool 播放音效……536
11.1.4 使用 VideoView 播放视频……539
11.1.5 使用 MediaPlayer 和 SurfaceView 播放视频……540
11.2 使用 MediaRecorder 录制音频……543
11.3 控制摄像头拍照……546
11.3.1 通过 Camera 进行拍照……546
11.3.2 录制视频短片……551
11.4 本章小结……555
第 12 章 OpenGL 与 3D 应用开发……556
12.1 3D 图像与 3D 开发的基本知识……557
12.2 OpenGL 和 OpenGL ES 简介……558
12.3 绘制 2D 图形……559
12.3.1 在 Android 应用中使用 OpenGL ES……559
12.3.2 绘制平面上的多边形……562
12.3.3 旋转……567

12.4 绘制 3D 图形……569
12.4.1 构建 3D 图形……569
12.4.2 应用纹理贴图……573
12.5 本章小结……578
第 13 章 Android 网络应用……579
13.1 基于 TCP 协议的网络通信……580
13.1.1 TCP 协议基础……580
13.1.2 使用 ServerSocket 创建 TCP 服务器端……581
13.1.3 使用 Socket 进行通信……582
13.1.4 加入多线程……586
13.2 使用 URL 访问网络资源……592
13.2.1 使用 URL 读取网络资源……593
13.2.2 使用 URLConnection 提交请求……594
13.3 使用 HTTP 访问网络……599
13.3.1 使用 HttpURLConnection……600
13.3.2 使用 Apache HttpClient……605
13.4 使用 WebView 视图显示网页……609
13.4.1 使用 WebView 浏览网页……610
13.4.2 使用 WebView 加载 HTML 代码……611
13.4.3 使用 WebView 中的 JavaScript 调用 Android 方法……612
13.5 使用 Web Service 进行网络编程……615
13.5.1 Web Service 平台概述……615
13.5.2 使用 Android 应用调用 Web Service……617
13.6 本章小结……628
第 14 章 管理 Android 手机桌面……629
14.1 管理手机桌面……630
14.1.1 删除桌面组件……630
14.1.2 添加桌面组件……630
14.2 改变手机壁纸……631
14.2.1 开发动态壁纸（Live Wallpapers）……631
14.3 通过程序添加快捷方式……636
14.4 管理桌面控件……638
14.4.1 开发桌面控件……638
14.4.2 Android 4.0 新增的显示数据集的桌面控件……642
14.5 本章小结……647
第 15 章 传感器应用开发……648
15.1 利用 Android 的传感器……649
15.1.1 开发传感器应用……649
15.2 Android 的常用传感器……651
15.2.1 方向传感器 Orientation……651
15.2.2 磁场传感器 Magnetic Field……652
15.2.3 温度传感器 Temperature……652
15.2.4 光传感器 Light……652
15.2.5 压力传感器 Pressure……653
15.3 传感器应用案例……655
15.4 本章小结……660
第 16 章 GPS 应用开发……661
16.1 支持 GPS 的核心 API……662
16.2 获取 LocationProvider……664
16.2.1 获取所有可用的 LocationProvider……664
16.2.2 通过名称获得指定 LocationProvider……665
16.2.3 根据 Criteria 获得 LocationProvider……665
16.3 获取定位信息……666
16.3.1 通过模拟器发送 GPS 信息……666
16.3.2 获取定位数据……667
16.4 临近警告……668
16.5 本章小结……670
第 17 章 使用 Google Map 服务……671
17.1 调用 Google Map 的准备……672
17.1.1 获取 Map API Key……672
17.1.2 创建支持 Google Map API 的 AVD……674
17.2 根据 GPS 信息在地图上定位……676
17.3 GPS 导航……681
17.4 根据地址定位……683
17.4.1 地址解析与反向地址解析……683
17.4.2 根据地址定位……688
17.5 本章小结……689

第 18 章　疯狂连连看 ……690
18.1　连连看游戏简介 ……691
18.2　开发游戏界面 ……691
18.2.1　开发界面布局 ……692
18.2.2　开发游戏界面组件 ……693
18.2.3　处理方块之间的连接线 ……696
18.3　连连看的状态数据模型 ……697
18.3.1　定义数据模型 ……697
18.3.2　初始化游戏状态数据 ……698
18.4　加载界面的图片 ……700
18.5　实现游戏 Activity ……703
18.6　实现游戏逻辑 ……708
18.6.1　定义 GameService 组件接口 ……708
18.6.2　实现 GameService 组件 ……709
18.6.3　获取触碰点的方块 ……710
18.6.4　判断两个方块是否可以相连 ……711
18.6.5　定义获取通道的工具方法 ……713
18.6.6　没有转折点的横向连接 ……715
18.6.7　没有转折点的纵向连接 ……715
18.6.8　一个转折点的连接 ……716
18.6.9　两个转折点的连接 ……718
18.6.10　找出最短距离 ……724
18.7　本章小结 ……726
第 19 章　电子拍卖系统 ……727
19.1　系统功能简介和架构设计 ……728
19.1.1　系统功能简介 ……728
19.1.2　系统架构设计 ……729
19.2　JSON 简介 ……730
19.2.1　使用 JSON 语法创建对象 ……731
19.2.2　使用 JSON 语法创建数组 ……732
19.2.3　Java 的 JSON 支持 ……733
19.3　发送请求的工具类 ……734
19.4　用户登录 ……735
19.4.1　处理登录的 Servlet ……736
19.4.2　用户登录 ……737
19.5　查看流拍物品 ……745
19.5.1　查看流拍物品的 Servlet ……745
19.5.2　查看流拍物品 ……746
19.6　管理物品种类 ……751
19.6.1　浏览物品种类的 Servlet ……752
19.6.2　查看物品种类 ……752
19.6.3　添加种类的 Servlet ……757
19.6.4　添加物品种类 ……758
19.7　管理拍卖物品 ……760
19.7.1　查看自己的拍卖物品的 Servlet ……760
19.7.2　查看自己的拍卖物品 ……761
19.7.3　添加拍卖物品的 Servlet ……764
19.7.4　添加拍卖物品 ……765
19.8　参与竞拍 ……771
19.8.1　选择物品种类 ……771
19.8.2　根据种类浏览物品的 Servlet ……772
19.8.3　根据种类浏览物品 ……773
19.8.4　参与竞价的 Servlet ……775
19.8.5　参与竞价 ……776
19.9　权限控制 ……781
19.10　本章小结 ……782

CHAPTER

1

第 1 章
Android 应用与开发环境

本章要点

- Android 手机平台的发展与现状
- Android 手机平台的架构与特性
- 搭建 Android 应用的开发环境
- 管理 Android 虚拟设备
- 使用 Android 模拟器
- 调试工具 DDMS 的用法
- 使用 ADB 工具复制文件、安装 APK 等
- DX、AAPT 工具的用法
- 在 Eclipse 中使用 ADT 开发 Android 应用
- 手动开发 Android 应用
- 掌握 Android 应用的结构
- 自动生成 Android 应用的清单文件
- Android 应用的 res 目录
- Android 应用的程序权限
- Android 应用的四大组件
- 对 Android 应用程序进行签名

Android 系统已经成为全球应用最广泛的手机操作系统，三星、摩托罗拉、HTC 等手机厂商早已通过 Android 阵营取得了巨大成功。目前国内对 Android 开发人才的需求也在迅速增长。而且搭载 Android 智能系统的手机越来越不像“手机”，更像是一台小型计算机。因此手机软件必将在未来 IT 行业中具有举足轻重的地位——你不可能带着一台电脑到处跑，而且时时开着机，但手机可以做到。从趋势上来看，Android 软件人才的需求会越来越大。

本书所介绍的平台是 Android 4.2 平台，该版本的 Android 平台经过几年的沉淀，不仅功能十分强大，而且十分高效、稳定。本书将会全面介绍 Android 平台的软件开发。但本章是全书的基础，将会简要介绍 Android 平台的历史、现状，重点向读者讲解如何搭建和使用 Android 应用开发环境，包括安装 Android SDK、Android 开发工具；也包括如何使用 Android 提供的 ADB、DDMS、AAPT、DX 等工具，掌握这些工具是开发 Android 应用的基础技能。

1.1 Android 的发展和历史

Android 是由 Andy Rubin 创立的一个手机操作系统，后来被 Google 收购。Google 希望与各方共同建立一个标准化、开放式的移动电话软件平台，从而在移动产业内形成一个开放式的操作平台。

➤➤1.1.1 Android 的发展和简介

Android 并不是 Google 创造的，而是由 Android 公司所创造的，该公司的创始人是 Andy Rubin。该公司后来被 Google 收购，而 Andy Rubin 也成为 Google 公司的 Android 产品负责人。

Google 于 2007 年 11 月 5 日发布了 Android 1.0 手机操作系统，这个版本的 Android 系统还没有赢得广泛的市场支持。

2009 年 5 月，Google 发布了 Android 1.5，该版本的 Android 提供了一个非常“豪华”的用户界面，而且提供了蓝牙连接支持。这个版本的 Android 吸引了大量开发者的目光。接下来，Android 的版本更新得较快，目前最新的 Android 版本是 4.2，这也是本书所介绍的 Android 版本。

目前 Android 已经成为一个重要的手机操作系统。当前市场上常见的手机操作系统有如下这些。

- iOS：Apple 公司的手机、平板操作系统，市场占有率较高。
- Windows Phone：Microsoft 公司的手机操作系统，2012 年发布的最新版本为 Windows Phone 8，但局势依然不够明朗。
- Symbian：已被放弃、基本被淘汰。
- BlackBerry：即将被淘汰。

目前 Android 系统的市场占有率已经远超 iOS，而 Windows Phone 作为 Microsoft 最后的“赌注”，自然也是全力以赴，希望至少能与 iOS、Android 三足鼎立，但目前局势似乎并不乐观。无论从哪个角度来看，Android 已经成为最主流的手机操作系统。

就目前国内环境来说，已有大量手机厂商开始生产 Android 操作系统的手机，因为 Android 手机平台是一个真正开放式的平台，无须支付任何费用即可使用。出于节省研发费用的考虑，不管是对于知名手机生产厂商，还是大量的山寨手机厂商，Android 操作平台都是一个不错的选择。

从 2008 年 9 月 22 日，T-Mobile 在纽约正式发布第一款 Android 手机——T-Mobile G1 开始，Android 系统不断地获得各个手机厂商的青睐。

2010 年 1 月 7 日，Google 在其美国总部正式向外界发布了旗下首款合作品牌手机 Nexus One（HTC G5），同时开始对外发售。

目前，已发布搭载 Android 系统的手机的厂商包括：摩托罗拉、三星、HTC、索尼爱立信、LG 等；国内厂商如华为、联想、中兴等也都开始发布搭载 Android 系统的手机。

1.1.2 Android 平台架构及特性

Android 系统的底层建立在 Linux 系统之上，该平台由操作系统、中间件、用户界面和应用软件 4 层组成，它采用一种被称为软件叠层（Software Stack）的方式进行构建。这种软件叠层结构使得层与层之间相互分离，明确各层的分工。这种分工保证了层与层之间的低耦合，当下层的层内或层下发生改变时，上层应用程序无须任何改变。

图 1.1 显示了 Android 系统的体系结构。

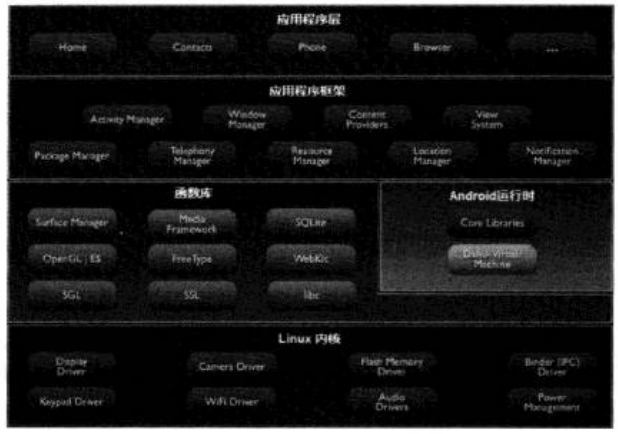


图 1.1 Android 系统的体系结构



备注： 图 1.1 直接引自 Android 官方文档。

从图 1.1 可以看出，Android 系统主要由 5 部分组成，下面分别对这 5 部分进行简单介绍。

1. 应用程序层

Android 系统将会包含系列的核心应用程序，包括电子邮件客户端、SMS 程序、日历、地图、浏览器、联系人等。这些应用程序都是用 Java 编写的。这也是本书所介绍的主要内容：编写 Android 系统上的应用程序。

2. 应用程序框架

前面已经提到，本书所要介绍的内容就是开发 Android 应用程序，当我们开发 Android 应用程序时，就是面向底层的应用程序框架进行的。从这个意义上来看，Android 系统上的应用程序是完全平等的，不管是 Android 系统提供的程序，还是普通开发者提供的程序，都可以访问 Android 提供的 API 框架。

Android 应用程序框架提供了大量 API 供开发者使用，关于这些 API 的具体功能和用法

则是本书后面要详细介绍的内容，此处不再展开阐述。

应用程序框架除可作为应用程序开发的基础之外，也是软件复用的重要手段，任何一个应用程序都可发布它的功能模块——只要发布时遵守了框架的约定，那么其他应用程序也可使用这个功能模块。

3．函数库

Android 包含一套被不同组件所使用的 C/C++库的集合。一般来说，Android 应用开发者不能直接调用这套 C/C++库集，但可以通过它上面的应用程序框架来调用这些库。

下面列出一些核心库。

- 系统 C 库：一个从 BSD 系统派生出来的标准 C 系统库（libc），并且专门为嵌入式 Linux 设备调整过。
- 媒体库：基于 PacketVideo 的 OpenCORE，这套媒体库支持播放和录制许多流行的音频和视频格式，以及查看静态图片。主要包括 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG 等多媒体格式。
- Surface Manager：管理对显示子系统的访问，并可以对多个应用程序的 2D 和 3D 图层机提供无缝整合。
- LibWebCore：一个全新的 Web 浏览器引擎，该引擎为 Android 浏览器提供支持，也为 WebView 提供支持，WebView 完全可以嵌入开发者自己的应用程序中。本书后面会有关于 WebView 的介绍。
- SGL：底层的 2D 图形引擎。
- 3D libraries：基于 OpenGL ES 1.0 API 实现的 3D 系统，这套 3D 库既可使用硬件 3D 加速（如果硬件系统支持），也可使用高度优化的软件 3D 加速。
- FreeType：位图和向量字体显示。
- SQLite：供所有应用程序使用的、功能强大的轻量级关系数据库。

4．Android 运行时

Android 运行时由两部分组成：Android 核心库集和 Dalvik 虚拟机。其中核心库集提供了 Java 语言核心库所能使用的绝大部分功能，而虚拟机则负责运行 Android 应用程序。



提示：

Android 运行时（是不是可以简称为 ARE？似乎还没有见到这种简称）和 JRE 有点类似。就像疯狂 Java 体系的《疯狂 Java 讲义》一书中解释的，JRE 包括 JVM 和其他功能函数库；而此处的 Android 运行时则包括 Dalvik 虚拟机和核心库集。

每个 Android 应用程序都运行在单独的 Dalvik 虚拟机内（即每个 Android 应用程序对应一条 Davlik 进程），Dalvik 专门针对同时高效地运行多个虚拟机进行了优化，因此 Android 系统以方便的实现对应用程序进行隔离。

由于 Android 应用程序的编程语言是 Java，因此有些人会把 Dalvik 虚拟机和 JVM 搞混，但实际上二者存在区别：Dalvik 并未完全遵守 JVM 规范，两者也不兼容。实际上，JVM 虚拟机运行的是 Java 字节码（通常就是.class 文件），但 Dalvik 运行的是其专有的 dex（Dalvik Executable）文件。JVM 直接从.class 文件或 JAR 包中加载字节码然后运行；而 Dalvik 则无法直接从.class 文件或 JAR 包中加载字节码，它需要通过 DX 工具将应用程序的所有.class 文

件编译成.dex 文件，Dalvik 则运行该.dex 文件。

Dalvik 虚拟机非常适合在移动终端上使用，相对于在 PC 或服务器上运行的虚拟机而言，Dalvik 虚拟机不需要很快的 CPU 计算速度和大量的内存空间，它主要有如下两个特点。

- 运行专有的.dex 文件。专有的.dex 文件减少了.class 文件中的冗余信息，而且会把所有.class 文件整合到一个文件中，从而提高运行性能；而且 DX 工具还会对.dex 文件进行一些性能的优化。
- 基于寄存器实现。大多数虚拟机（包括 JVM）都是基于栈的，而 Dalvik 虚拟机则是基于寄存器的。一般来说，基于寄存器的虚拟机具有更好的性能表现，但在硬件通用性上略差。

Dalvik 虚拟机依赖于 Linux 内核提供的核心功能，如线程和底层内存管理。

5. Linux 内核

Android 系统建立在 Linux 2.6 之上。Linux 内核提供了安全性、内存管理、进程管理、网络协议栈和驱动模型等核心系统服务。除此之外，Linux 内核也是系统硬件和软件叠层之间的抽象层。

1.2 搭建 Android 开发环境

在开始搭建 Android 开发环境之前，笔者假定读者已经具有一定的 Java 编程基础，像 JDK 安装、环境设置之类的入门知识不在本书介绍范围之内。如果读者暂时还不会这些知识，建议先学习 Java 入门知识。

下面将从 Android SDK 的安装开始讲起，详细说明 Android 开发、调试环境的安装和使用，这些内容是 Android 开发的基础。

1.2.1 下载和安装 Android SDK

Android 的官方站点是 http://www.android.com，登录该站点即可下载 Android SDK。下载和安装 Android SDK 请按如下步骤进行。

登录 http://developer.android.com/sdk/index.html 页面，点击最下方的 DOWNLOAD FOR OTHER PLATFORMS 链接，即可看到如图 1.2 所示的下载链接。



图 1.2 Android SDK 的下载链接

注意：

笔者使用电信 ADSL 无法正常访问上面的站点（该站点可能被电信封了，原因未知），如果读者也遇到这样的问题，建议设置代理服务器来访问该站点。





提示：

图 1.2 上方还可以看到了 3 个 ADT Bundle 的链接，单击该链接将会为不同平台下载 ADT Bundle 包，该包内包括两个文件夹：eclipse 和 sdk。其中 eclipse 文件夹内是一个已安装了 ADT 插件的 Eclipse；sdk 文件夹内就是此处下载的 SDK。如果读者以前没有任何 Java 开发经验，没有安装过 Eclipse，可以选择下载该压缩包，直接使用该压缩包内的 Eclipse，这个 Eclipse 已自带 ADT 插件，因此无须安装 ADT 插件。

② 单击图 1.2 所示页面上的“android-sdk_r21-windows.zip”链接，通过该链接即可下载 Android 4.2 SDK 压缩包。



提示：

不同平台下载相应的 SDK 压缩包即可，不管是 Windows 平台还是其他平台，都只需将下载得到的压缩包解压，再配置一些环境变量即可。

③ 下载完成后得到一个 android-sdk_r21-windows.zip 文件，将该文件解压缩到任意路径下，比如 D:\盘的根路径。解压缩后得到一个 android-sdk-windows 文件夹，该文件夹下包含如下文件结构。

- add-ons：该目录下存放第三方公司为 Android 平台开发的附加功能系统。刚解压缩时该目录为空。
- platforms：该目录下存放不同版本的 Android 系统。刚解压缩时该目录为空。
- tools：该目录下存放了大量 Android 开发、调试的工具。
- AVD Manager.exe：该程序是 AVD（Android 虚拟设备）管理器。通过该工具可以管理 AVD。
- SDK Manager.exe：该程序就是 Android SDK 管理器。通过该工具可以管理 Android SDK。

④ 启动 SDK Manager.exe，即可看到如图 1.3 所示窗口。

⑤ 在图 1.3 所示窗口的列表中勾选需要安装的平台和工具，比如 Android 4.2 的工具和平台，其中 Android 文档、SDK Platform 是必选的，如果想查看 Android 官方提供的示例程序、使用 Android SDK 的源代码，则可以勾选“Samples for SDK”和“Sources for Android SDK”两个列表项。至于是否需要安装 Android 早期版本的 SDK，则取决于读者喜好。选中所需安装的工具之后，单击“Install Selected”按钮，将看到出现如图 1.4 所示窗口。

⑥ 单击图 1.4 所示窗口中的“Accept All”单选按钮——确认需要安装所有的工具包。然后单击“Install”按钮，系统开始在线安装 Android SDK 及相关工具。取决于读者的网络状态及选中的工具包的数量，在线安装时间不会太短，甚至可能花费一两个小时，耐心等待即可。



图 1.3 Android SDK 和 AVD 管理器

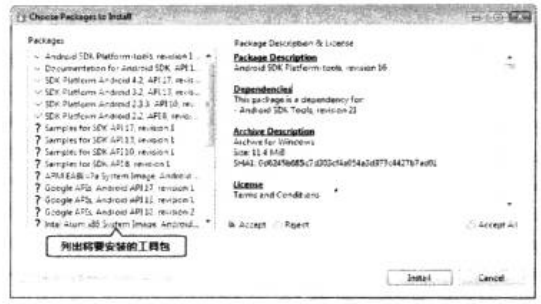


图 1.4 列出将要安装的 Android 工具包

⑦ 安装完成后将可以看到在 Android SDK 目录下增加了如下几个文件夹。

- docs：该文件夹下存放了 Android SDK 开发文件和 API 文档等。
- extras：该文件夹下存放了 Google 提供的 USB 驱动、Intel 提供的硬件加速等附加工具包。
- platform-tools：该文件夹下存放了 Android 平台相关工具。
- samples：该文件夹下存放了不同 Android 平台的示例程序。
- sources：该文件夹下存放了 Android 4.2 的源代码。

⑧ 为了在命令行窗口可以使用 Android SDK 的各种工具，建议将 Android SDK 目录下的 tools 子目录、platform-tools 子目录添加到系统的 PATH 环境变量中。

1.2.2 安装运行、调试环境

Android 程序必须在 Android 手机上运行，因此 Android 开发时必须准备相关运行、调试环境。准备 Android 程序的运行、调试环境有如下两种方式。

- 条件允许，优先考虑购买 Android 真机（真机调试的速度更快、效果更好）。
- 配置 Android 虚拟设备（即 AVD）。

1. 使用真机作为运行、调试环境

使用真机作为运行、调试环境时，只要完成如下 3 步。

① 用 USB 连接线将 Android 手机连接到电脑上。

② 在电脑上为手机安装驱动，不同手机厂商的 Android 手机的驱动略有差异，请登录各手机厂商官网下载手机驱动。

注意：

通常都需要在电脑上为手机安装驱动。可能有读者会感到疑惑：Android 手机连接电脑后，电脑即可识别到 Android 的存储卡，不需要安装驱动啊？需要提醒读者的是，电脑仅能识别 Android 手机的存储卡是不够的，安装驱动才能把 Android 手机整合成运行、调试环境。



③ 打开手机的调试模式。打开手机，依次单击“所有应用→设置→开发者选项”，进入如图 1.5 所示的设置界面。

按图 1.5 所示，勾选“不锁定屏幕”、“USB 调试”、“允许模拟位置”3 个选项即可。如果开发者还有其他需要，也可以勾选其他的开发者选项。

2. 使用 AVD 作为运行、调试环境

Android SDK 为开发者提供了可以在电脑上运行的“虚拟手机”，Android 把它称为 Android Virtual Device（AVD）。如果开发者没有 Android 手机，则完全可以在 AVD 上运行我们编写的 Android 应用。

创建、删除和浏览 AVD 之前，通常应该先为 Android SDK 设置一个环境变量：ANDROID_SDK_HOME，该环境变量的值为磁盘上一个已有的路径。如果不设置该环境变量，开发者创建的虚拟设备默认保存在 C:\Documents and Settings\<user_name>\.android 目录（以 Windows XP 为例）下；如果设置了 ANDROID_SDK_HOME 环境变量，那么虚拟设备就会保存在%ANDROID_SDK_ HOME%/.android 路径下。



图 1.5　打开调试模式

> **注意：**
> 这里有一点非常容易混淆，此处的%ANDROID_SDK_HOME%环境变量并不是 Android SDK 的安装目录。而学习过 Java EE 的读者可能都记得 JAVA_HOME、ANT_HOME 等环境变量，它们都是指向自身的安装目录，但 Android 的%ANDROID_ SDK_HOME%不是。



在图形界面下管理 AVD 比较简单，因为可以借助于 Android SDK 和 AVD 管理器完成，完全可以在图形用户界面下操作，比较适合新上手的用户。

① 通过 Android SDK 安装目录下 AVD Manager.exe 启动 AVD 管理器，系统启动如图 1.6 所示的 AVD 管理器。单击该管理器左边的“Virtual devices”项，管理器列出当前已有的 AVD 设备，如图 1.6 所示。

② 单击图 1.6 所示窗口右边的“New...”按钮，AVD 管理器弹出如图 1.7 所示对话框。

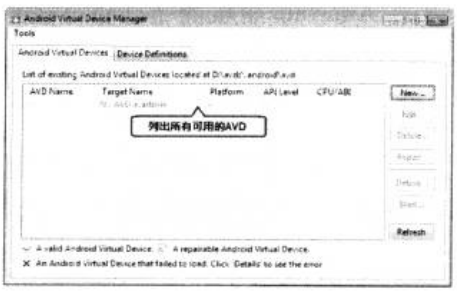


图 1.6　查看所有可用的 AVD 设备



图 1.7　创建 AVD 设备

③ 在图 1.7 所示的对话框中填写 AVD 设备的名称、Android 平台的版本和虚拟 SD 卡的大小，然后单击该对话框下面的“OK”按钮，管理器即将开始创建 AVD 设备，开发者只要

稍作等待即可。



提示：

Android 4.2 创建的 AVD 的默认分辨率是 WVGA800（即 800 × 480 的分辨率），对于配置不太好的电脑而言，选择 WVGA800 可能会导致虚拟机运行速度较慢的后果，因此笔者选择 HVGA（即 480 × 320 的分辨率）。

创建完成后，管理器返回如图 1.6 所示的窗口，该管理器将会列出当前所有可用的 AVD 设备。如果开发者想删除某个 AVD 设备，只要在图 1.6 所示窗口中选择指定 AVD 设备，然后单击右边的“Delete...”按钮即可。

AVD 设备创建成功之后，接下来就可以使用模拟器来运行该 AVD 了。在 Android SDK 和 AVD 管理器中运行 AVD 非常简单：① 在图 1.6 所示窗口中选中需要运行的 AVD 设备；② 单击图 1.6 所示窗口中的“Start...”按钮即可。启动后的虚拟手机如图 1.8 所示。

图 1.8 就是一个运行在电脑上的虚拟手机，用过手机的读者对这个界面应该不会陌生。现在已经开始使用该“虚拟手机”来模拟一些“手机操作”，读者可以花点时间来熟悉一下 Android 系统的操作习惯。

单击 Android 桌面上的“访问程序”按钮，Android 进入如图 1.9 所示的界面。



图 1.8　启动后的虚拟手机



图 1.9　Android 应用程序列表

从图 1.9 所示列表中可以看到 Android 系统默认提供的所有可用的程序，以后我们开发的 Android 程序也可以在这里找到。当包含的程序太多时，可以通过手指左右拖动来查看更多的程序。

对于国内用户来说，设置中文操作界面、设置中文输入法是两个常用的操作。设置中文操作界面可通过单击图 1.9 所示界面中的“Settings”项来进行，依次单击“Settings→Language & input→Language”，Android 系统将出现如图 1.10 所示列表，选中其中的“中文（简体）”列表项，然后单击虚拟手机上的确认键返回。



注意：

为 Android 系统设置了中文操作界面之后，在有些电脑上启动、运行模拟器特别慢，慢到令人难以忍受。如果遇到这种情况，请放弃使用中文操作界面。





图 1.10　设置中文操作界面

开启中文输入法通过单击图 1.9 所示界面中“Settings”项进行设置，依次单击“Settings→Language & input”，在出现的列表中勾选“谷歌拼音输入法”列表项，然后单击虚拟手机上的确认键返回即可。



提示：

有时开发者启动了 Android 模拟器，“虚拟手机”的显示屏右上方可能提示没有网络信号，通常是因为模拟器无法访问网络的缘故。一般来说，只要运行模拟器的电脑已经处于局域网内（已接入 Internet 也可以），并且没有防火墙阻止 Android 模拟器访问网络，Android 模拟器都不应该提示没有网络信号。如果运行 Android 模拟器的机器既不在局域网内，也没有接入 Internet，则可将电脑 DNS 服务器设为与本机相同。例如设置本机 IP 为 192.168.1.50，再将 DNS 服务器地址也设为 192.168.1.50 即可。

1.2.3　安装 Eclipse 和 ADT 插件

Eclipse 是一个市场占有率很高的 Java IDE 工具，Java EE 应用的开发者对 Eclipse 应该十分熟悉。Android 推荐使用 Eclipse 来开发 Android 应用，并为它提供了专门的插件：Android Development Tools（ADT）。

安装 Eclipse 工具按如下步骤进行。

① 登录 http://www.eclipse.org 站点，下载 Eclipse IDE for Java EE Developers 的最新版本。该版本 Eclipse 当前的最新版本是 Eclipse-jee-juno-SR1 版（也就是 Eclipse 4.2），笔者使用的正是该版本的 Eclipse。



提示：

Eclipse IDE for Java EE Developers 是 Eclipse 为 Java EE 开发者准备的一个 IDE 工具，它在“纯净”Eclipse 的基础之上，集成了一些 Eclipse 插件，允许开发者不需额外添加插件即可进行 Java EE 开发。

② Windows 平台下载 eclipse-jee-juno-SR1-win32.zip 文件，Linux 平台下载 eclipse-jee-juno-SR1-linux-gtk.tar.gz 文件。解压缩下载得到的压缩文件，解压后的文件夹可放在任何目录。

③ 直接双击 eclipse.exe 文件，即可看到 Eclipse 的启动界面，表明 Eclipse 已经安装成功。

为了在 Eclipse 中进行 Android 开发，还需要安装 ADT 插件。安装 ADT 请按如下步骤进行。

① 登录 http://developer.android.com/sdk/installing/installing-adt.html，下载 ADT 插件的最新版本。ADT 的最新版本为 21.0.0.zip，笔者使用的是该版本的 ADT。

② 下载完成后得到一个 ADT-21.0.0.zip 文件，它就是一个 Eclipse 插件。

③ 启动 Eclipse，单击 Eclipse 主菜单上的 Help→Install New Software...菜单项，如图 1.11 所示。

④ Eclipse 弹出如图 1.12 所示的窗口。

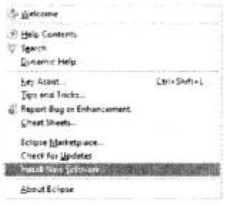


图 1.11 安装 Eclipse 插件

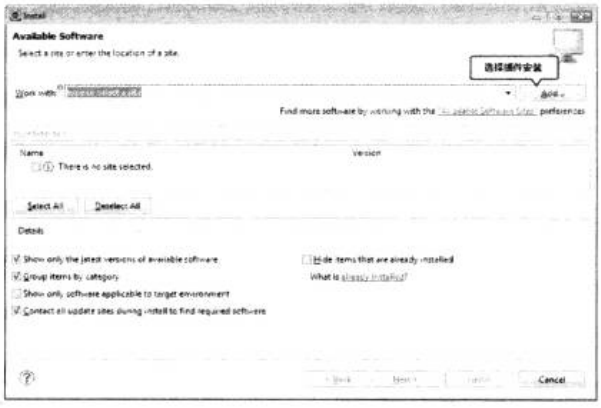


图 1.12 选择插件安装

⑤ 单击图 1.12 所示窗口的“Add...”按钮，Eclipse 弹出如图 1.13 所示的对话框。

⑥ 在图 1.13 所示对话框中通过“Archive...”按钮选择 ADT 插件的 ADT-21.0.0.zip 文件。选中后单击“OK”按钮，Eclipse 返回图 1.12 所示的窗口，但此处列出了可用的 ADT 插件，如图 1.14 所示。



图 1.13 浏览 ADT 插件

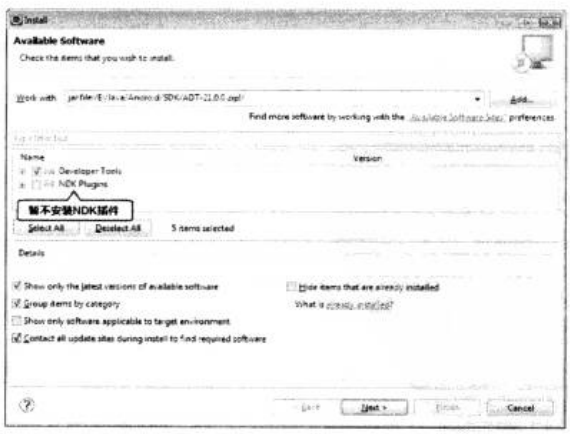


图 1.14 勾选 ADT 插件准备安装

⑦ 在图 1.14 所示对话框中勾选 Developer Tools 项（即 ADT 插件）项，然后单击“Next”按钮，Eclipse 弹出一个对话框，该对话框提示用户所有将要安装的插件详细清单，单击该对话框的“Next”按钮，Eclipse 将弹出如图 1.15 所示的窗口。

注意：

在该步骤中，如果不是立即需要使用NDK开发，可以先不勾选NDK Plugins项。如果勾选、安装该插件项，由于该插件还需要依赖C、C++开发插件，Eclipse将需要联网下载C、C++开发插件。

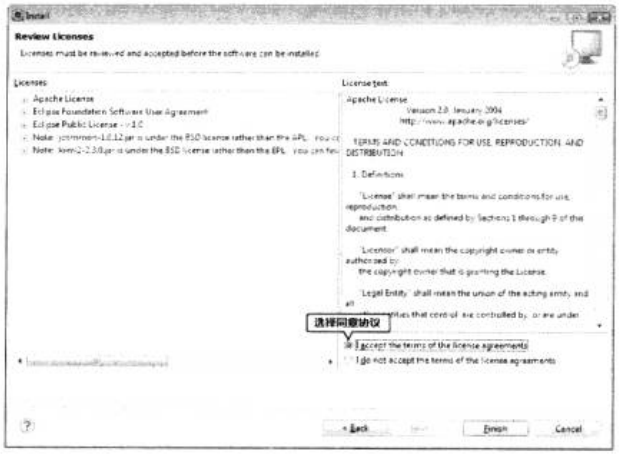


图 1.15　选择同意协议

⑧ 在图 1.15 所示的窗口中选择接受协议，然后单击“Finish”按钮，Eclipse 开始安装 ADT 插件。稍稍等待几分钟，ADT 插件安装完成。

⑨ ADT 插件安装完成后，Eclipse 会通知开发者需要重启。重启 Eclipse 会弹出如图 1.16 所示的窗口。

为 Eclipse 安装了 ADT 插件之后，在 Eclipse 工具条上将看到增加了如图 1.17 所示的 4 个按钮。



图 1.16　设置 Android SDK 的安装目录



图 1.17　ADT 插件的按钮

图 1.17 中前两个按钮只是 Android SDK 目录下 SDK Manager.exe 和 AVD Manager.exe 工具的快捷方式。

提示：

如果在图 1.16 所示窗口中没有正确设置 Android SDK 的安装目录，也可以直接在 Eclipse 中设置 Android SDK 的路径。单击 Eclipse 主菜单 Window→Preferences 菜单项，Eclipse 弹出如图 1.18 所示窗口。

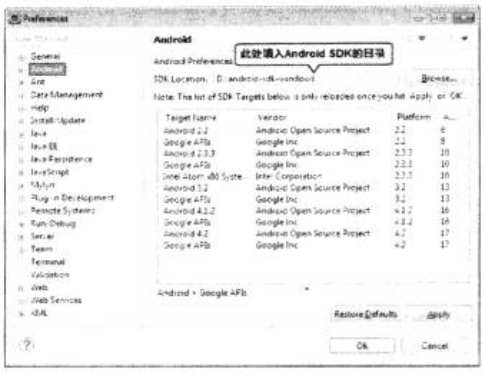


图 1.18 设置 Android SDK 路径

在图 1.18 所示窗口的文本框中填入 Android SDK 的安装目录。

经过上面所介绍的过程，接下来就可以在 Eclipse 中开发 Android 应用了。

1.3 Android 常用开发工具的用法

前面主要介绍了 Android SDK 的安装，运行、调试环境的的搭建，以及 Android 开发环境 Eclipse 和 ADT 插件的安装，但这些内容只是最基本的知识，要真正掌握 Android 开发，还必须掌握 Android 开发的大量辅助工具。

1.3.1 在命令行创建、删除和浏览 AVD

在命令行下管理 AVD 需要借助于 android 命令（位于 Android SDK 安装目录的 tools 子目录下），如果直接执行 android 命令将会启动 Android SDK 管理器。除此之外，该命令还支持如下子命令。

- list：列出机器上所有已经安装的 Android 版本和 AVD 设备。
- list avd：列出机器上所有已经安装的 AVD 设备。
- list target：列出机器上所有已经安装的 Android 版本。
- create avd：创建一个 AVD 设备。
- move avd：移动或重命名一个 AVD 设备。
- delete avd：删除一个 AVD 设备。
- update avd：升级一个 AVD 设备使之符合新的 SDK 环境。
- create project：创建一个新的 Android 项目。
- update project：更新一个已有的 Android 项目。
- create test-project：创建一个新的 Android 测试项目。
- update test-project：更新一个已有的 Android 测试项目。

如果希望查看当前系统上已安装的 Android 版本及已安装的 AVD 设备，则运行 android list 或者 android list avd 命令即可，如图 1.19 所示。

如果要创建一个全新的 AVD 设备，可执行如下命令：

```
android create avd -n <avd名称> -t <Android版本> -b <CPU架构>
-p <AVD设备保存位置> -s <选择AVD皮肤>
```

在上面的 create avd 子命令中，只有-n 和-t 选项是必需的，其余的-b 选项、-p 选项、-t 选项都是可选的。如果不设置-p 选项，创建的 AVD 设备默认保存在%ANDROID_SDK_HOME%/.android/avd 路径下。

例如需要创建一个名为 leegang 的 AVD 设备，则可输入如下命令：

```
android create avd -n crazyit -t 10 -b armeabi-v7a
```

上面的命令中 10 是 Android 4.2 的代号，如图 1.19 所示。执行上面的命令，系统会提醒用户是否需要定制 AVD 的硬件，开发者可以选择 yes 或 no，如果输入 no，即可直接开始创建 AVD 设备；如果输入 yes，即可开始定制 AVD 硬件的各种选项，定制完成后系统开始创建 AVD 设备。

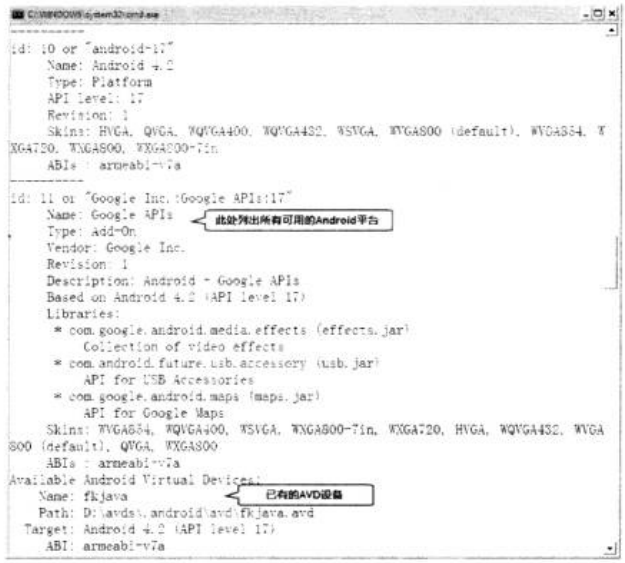


图 1.19　列出已经安装的 Android 版本和 AVD 设备

为系统创建了两个 AVD 之后（前面通过图形界面创建了一个），即可在%ANDROID_SDK_HOME%/.android 目录下看到一个 avd 子目录，该子目录下包含两个文件和两个文件夹。

- fkjava.avd 和 fkjava.ini：fkjava AVD 的基本信息和 AVD 设备。其中 fkjava.avd 目录下有一个 userdata.img 文件，它是 AVD 中用户数据的镜像。还有一个 sdcard.img，它是该 AVD 所使用的虚拟 SD 卡的镜像。
- crazyit.avd 和 crazyit.ini：crazyit AVD 的基本信息和 AVD 设备。其中 crazyit.avd 目录下有一个 userdata.img 文件，它是 AVD 中用户数据的镜像。

1.3.2　使用 Android 模拟器（Emulator）

Android 模拟器就是一台运行在电脑上的“虚拟手机”。实际上前面我们已经使用过 Android 模拟器了，在 Android SDK 和 AVD 管理器中选中指定 AVD 设备，然后单击“Start...”按钮就是启动模拟器来运行 Android 系统。

在 Android SDK 安装目录的 tools 子目录下有一个 emulator.exe（另外还有 emulator-arm.exe 和 emulator-x86.exe），它们都是 Android 模拟器。这个模拟器做得十分出色，几乎可以模拟真实手机的绝大部分功能，后面我们会陆续看到——当然它只是模拟，不要指望用模拟器

与你现实中的朋友“煲电话粥”。

使用 emulator.exe 启动模拟器有两种用法：

- emulator -avd <AVD 名称>
- emulator -data 镜像文件名称

第一种用法是运行指定的 AVD 设备，例如如下命令：

```
emulator -avd crazyit //运行名为 crazyit 的 AVD 设备
```

第二种用法是直接使用指定镜像文件来运行 AVD，例如如下命令：

```
emulator -data myfile //以 myfile 作为镜像文件来运行 AVD 设备
```

1.3.3 使用 DDMS 进行调试

当 Android 应用在模拟器上运行时，我们看不到程序运行的过程，在命令行控制台也看不到程序的输出，如何调试 Android 应用呢？

不用担心，Android 已经为我们考虑好了这个问题。Android 提供了一个 DDMS 调试环境，DDMS 的全称是 Dalvik Debug Monitor Service，它是一个功能非常强大的调试环境。运行如下命令：

```
ddms.bat
```

即可看到系统启动如图 1.20 所示窗口。



图 1.20 DDMS 的调试窗口

在图 1.20 所示窗口中有如下几个重要的面板。

- 设备面板：DDMS 窗口左上角的面板，该面板会列出当前所有运行的手机（包括真机和模拟器），并列出各手机内的所有进程信息。如果需要查看指定手机或指定进程信息，应先在该面板内选中指定手机或进程。
- 信息输出面板：该面板位于 DDMS 窗口的下方，相当于传统 Java 应用控制台，因此非常重要。
- 线程跟踪面板：该面板可用于查看指定进程内所有正在执行的线程的状态。如需要让该面板显示指定进程内线程的状态，应保证下面两步：① 在设备面板上按下“Show thread updates”按钮；② 在设备面板上选中需要查看的进程。

- Heap 内存跟踪面板：该面板可用于查看指定进程内堆内存的分配和回收信息。如需要让该面板显示指定进程内 Heap 的回收和分配状态，应保证下面两步：① 在设备面板上按下“Show heap updates”按钮；② 在设备面板上选中需要查看的进程。
- 模拟器控制面板：该面板用于让模拟器模拟拨打电话、发送短信等，还可以虚拟设置模拟器的位置信息等。图 1.21 显示了该面板的示意图。
- 文件管理对话框：该对话框默认并没有显示出来。读者可以通过单击 DDMS 窗口上主菜单“Device→File Explorer...”来打开如图 1.22 所示的文件管理窗口。



图 1.21　模拟器控制面板

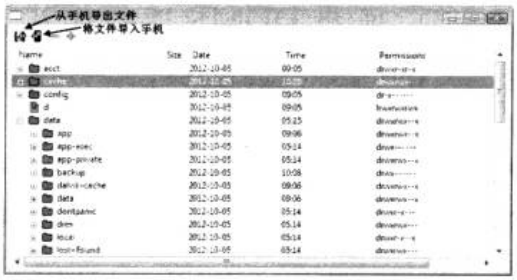


图 1.22　文件管理窗口

实际上，如果我们为 Eclipse 安装了 ADT 插件，那么 Eclipse 就会将 DDMS 集成进来，在 Eclipse 中可以直接切换到 DDMS 视图（Perspective）。要想将 Eclipse 切换到 DDMS 视图，只要按如下步骤操作。

① 单击 Eclipse 右上角的“Open Perspective”按钮，Eclipse 出现如图 1.23 所示菜单。

② 从图 1.23 所示的对话框中选择“DDMS”，然后单击“OK”按钮，即可将 Eclipse 切换到 DDMS 视图。

如果只是想在 Eclipse 中打开指定面板，并不想完全切换到 DDMS 视图，则可以单击 Eclipse 的主菜单的“Window→Show View”菜单项，系统出现如图 1.24 所示菜单。

在图 1.24 所示菜单中并没有看到任何 Android 相关的面板，可以单击“Other...”菜单项，系统打开如图 1.25 所示的窗口。



图 1.23　选择 DDMS 视图



图 1.24　打开指定面板



图 1.25　选择 Android 相关面板

1.3.4　Android Debug Bridge（ADB）的用法

Android Debug Bridge（ADB）是一个功能非常强大的工具，它位于 Android SDK 安装目

录的 platform-tools 子目录下。ADB 工具既可完成模拟器文件与电脑文件的相互复制，也可安装 APK 应用，甚至可以直接切换到 Android 系统中执行 Linux 命令。

ADB 工具的功能很多，此处就几个常用的功能略做说明。

1. 查看当前运行的模拟器

输入如下命令即可查看当前运行的模拟器：

```
adb -devices
```

2. 电脑与手机之间文件的相互复制

默认情况下，ADB 工具总是操作当前正在运行的模拟器。

如果需要将电脑文件复制到模拟器中，可使用 adb push 命令：

```
adb push d:/abc.txt /sdcard/
```

上面的命令将电脑的 D:\盘根目录下的 abc.txt 文件复制到手机的/sdcard/目录下。

如果需要将模拟器文件复制到电脑中，可使用 adb pull 命令：

```
adb pull /sdcard/xyz.txt d:/
```

上面的命令将模拟器的/sdcard/目录下的 xyz.txt 文件复制到电脑的 D:\盘根目录下。

3. 启动模拟器的 shell 窗口

Android 平台的内核是基于 Linux 的，有时开发者希望直接打开 Android 平台的 shell 窗口，这样就可以在该窗口内执行一些常用的 Linux 命令，如 ls、mkdir、rm 等。此时可考虑使用 adb shell 命令：

```
adb shell
```

4. 安装、卸载 APK 程序

APK 程序就是 Android 程序的发布包。虽然我们使用 Java 开发了 Android 应用，但并不是直接将 Java 二进制文件复制到手机或模拟器上即可。为了把 Android 应用打包成一个可发布的 APK 包，还需要经过如下 3 步。

① 通过 DX 工具对*.class 文件进行转换。转换后通常得到一个*.dex 文件。

② 通过 AAPT 工具打包所有的资源文件。打包后通常得到*.ap_文件。

③ 通过 apkbuilder 工具把前两步得到的*.dex、*.ap_文件打包成 APK 包。

一旦将 Android 应用打包成 APK 包，接下来就可以通过 ADB 工具来安装、卸载 APK 程序。

使用 ADB 安装 APK 的命令格式如下：

```
adb install [-r] [-s] <file>
```

上面的命令格式指定安装<file>代表的 APK 包。其中-r 表示重新安装该 APK 包；-s 表示将 APK 包安装到 SD 卡上——默认是将 APK 包安装到内部存储器上。例如如下命令即可安装 test.apk 包：

```
adb install text.apk
```

如果希望从 Android 系统中删除指定软件包，则可使用如下命令：

```
adb uninstall [-k] <package>
```

上面的命令格式指定删除<package>代表的 APK 包。其中-k 表示只删除该应用程序，但

保留该程序所用的数据和缓存目录。

1.3.5 使用 DX 编译 Android 应用

前面已经提到，Android 平台有一个重要的概念：Android 运行时。Android 运行时使用的虚拟机并没有遵循 JVM 规范，Android 所使用的虚拟机是 Dalvik 虚拟机。

Dalvik 虚拟机并不直接运行 Java 二进制文件，而是运行它特有的*.dex 文件，因此我们需要通过 DX 工具将 Android 应用的*.class 文件转换为*.dex 文件。

DX 工具的常见命令格式如下：

```
dx --dex [--dump-to=<file>] [--core-library] [<file>.class | <file>.{zip,jar,apk}
| <directory>]
```

上面的命令中[--dump-to=<file>]指定生成的*.dex 文件的文件名；而--core-library 指定需要转换的*.class、*.zip、*.jar 文件或者目录。

例如如下命令：

```
dx -dex --dump-to=g:\a.dex --core-library d:\helloworld\bin
```

将 d:\helloworld\bin 路径下所有二进制文件转换为 g:\根目录下的 a.dex 文件。

1.3.6 使用 Android Asset Packaging Tool（AAPT）打包资源

当开发 Android 应用时，该应用中可能会包含许多资源文件，包括各种图片、音频文件等。当我们需要发布一个 APK 包时，这些资源文件也是必不可少的。

AAPT 工具也支持很多子命令。

- aapt l[ist]：列出资源压缩包内的内容。
- aapt d[ump]：查看 APK 包内的指定内容。
- aapt p[ackage]：打包生成资源压缩包。
- aapt r[emove]：从压缩包中删除指定文件。
- aapt a[dd]：向压缩包中添加指定文件。
- aapt v[ersion]：打印 AAPT 的版本。

通过上面的介绍不难看出，应用使用 appt p 命令来打包资源包。APPT 工具打包资源包时常用的语法格式如下：

```
aapt -A <附件资源路径> -S <资源路径> -M <Android 应用清单文件> -I <额外添加的包> And -F
目标文件的路径
```

例如执行如下命令：

```
aap -A assets -S res -M AndroidManifest.xml
 -I D:\android-sdk-windows\platforms\android-9\atforms\android-9\android.jar -F
 bin\res.ap_
```

上面的命令将当前目录下 assets 子目录、res 子目录、AndroidManifest.xml 文件都打包到 bin\res.ap_资源包中。

1.3.7 使用 mksdcard 管理虚拟 SD 卡

正如前面在 Android SDK 和 AVD 管理中见到的，我们可以在创建 AVD 设备时创建一个

虚拟SD卡。实际上还可以使用mksdcard命令来单独创建一个虚拟存储卡。

mksdcard命令的语法格式如下：

```
mksdcard [-l label] <size> <file>
```

上面的命令格式中<size>指定虚拟SD卡的大小，<file>指定保存虚拟SD卡的文件镜像。例如如下命令：

```
mksdcard 64M D:\avds\.android\avd\leegang.avd\sdcard.img
```

创建了一个大小为64MB的虚拟SD卡，该SD卡对应的镜像文件为D:\avds\.android\avd\leegang.avd\sdcard.img。

如果希望在启动模拟器时使用指定虚拟SD卡，则在启动模拟器时增加-sdcard <file>选项，其中<file>代表了虚拟SD卡的文件镜像。例如如下命令：

```
emulator -avd crazyit -sdcard d:\sdcard.img
```

到此为止，我们已经成功地安装了Android SDK、配置了Android开发环境，并且对Android相关开发工具都有了一个大致的了解，接下来正式开始Android应用开发。

1.4 开始第一个Android应用

无须担心，Android应用的开发十分简单！Android应用程序建立在应用程序框架之上，所以Android编程就是面向应用程序框架API编程——这种开发方式与编写普通Java SE应用程序并没有太大的区别，只是Android新增了一些API而已。

1.4.1 使用Eclipse开发第一个Android应用

使用Eclipse开发Android应用非常方便，因为Eclipse会为我们自动完成许多工作。使用Eclipse开发Android应用大致需要如下3步。

1. 创建一个Android项目。
2. 在XML布局文件中定义应用程序的用户界面。
3. 在Java代码中编写业务实现。

上面3个步骤只是最粗粒度的归纳。下面以开发一个HelloWorld应用为例来介绍Android开发。详细步骤如下。

1. 通过Eclipse主菜单的“File→Other...”菜单项，创建一个Android项目。Eclipse弹出如图1.26所示窗口。

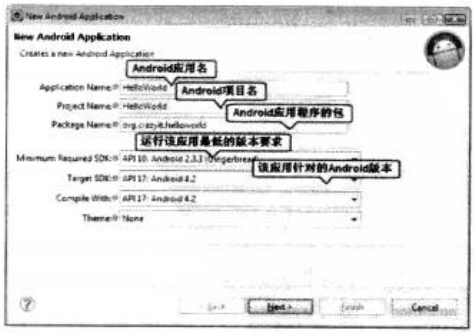


图1.26 创建Android项目

2. 在图1.26所示窗口中输入Android应用的项目名、应用程序名、包名，并选择Android应用针对的Android版本。单击图1.26所示窗口中的“Next”按钮。



提示：

Android应用程序的包名非常重要，Android应用的包名可作为Android应用的唯一标识。

③ 如果开发者没有勾选图 1.26 所示窗口的下一个窗口中“Create custom laucher icon”复选框，应用将会采用 Android SDK 默认提供的图标；如果勾选了“Create custom laucher icon”复选框，系统将会弹出如图 1.27 所示的窗口。

④ 按图 1.27 所示方式制作图标完成后，单击“Next”按钮，Eclipse 弹出如图 1.28 所示的窗口。

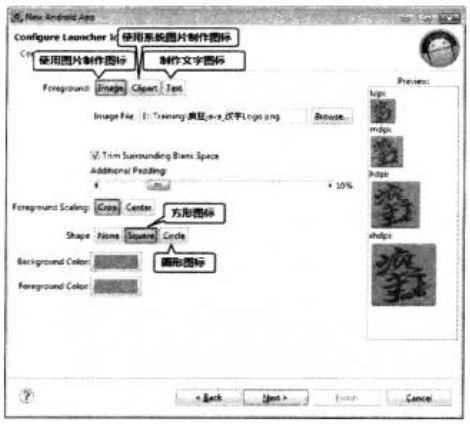


图 1.27　制作自定义图标

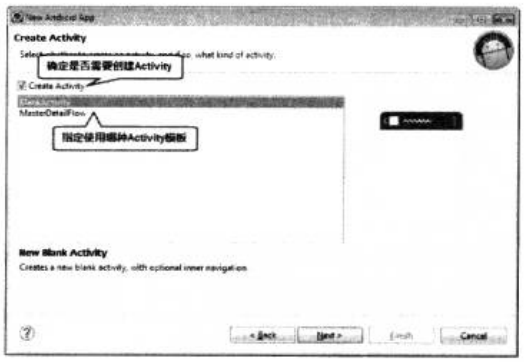


图 1.28　设置是否需要创建 Activity

⑤ 如果不勾选图 1.28 所示窗口的“Create Activity”复选框，即可单击该对话框中的“Finish”按钮来创建项目。如果勾选了“Create Activity”复选框，则只能单击“Next”按钮进入如图 1.29 所示的创建 Activity 的窗口。

⑥ 单击图 1.29 所示窗口中的“Finish”按钮，Eclipse 即成功创建了一个 Android 项目。Android 项目创建完成后将看到如图 1.30 所示项目结构。

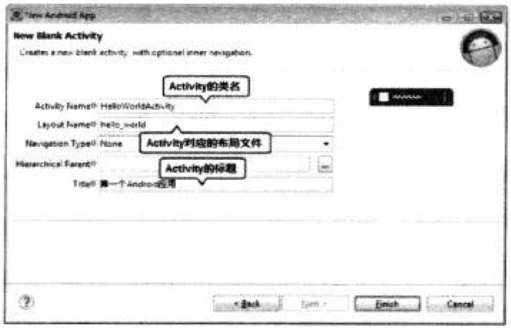


图 1.29　为创建 Activity 设置信息



图 1.30　Android 项目结构

⑦ Android 项目的 layout 目录下有一个 hello_world.xml 文件，该文件用于定义 Android 的应用用户界面。在 Eclipse 工具中打开该文件，将看到如图 1.31 所示界面。

图 1.31 所示的“所见即所得”的设计界面十分简单，有过网页编辑经验的读者可能对 Dreamweaver 之类的工具比较熟悉，那么可以把这个界面近似地当成 Dreamweaver。

在图 1.31 所示界面的控件面板中向程序拖入一个“Button”控件（按钮），再切换到源代码编写界面，将 main.xml 文件修改为如下形式。

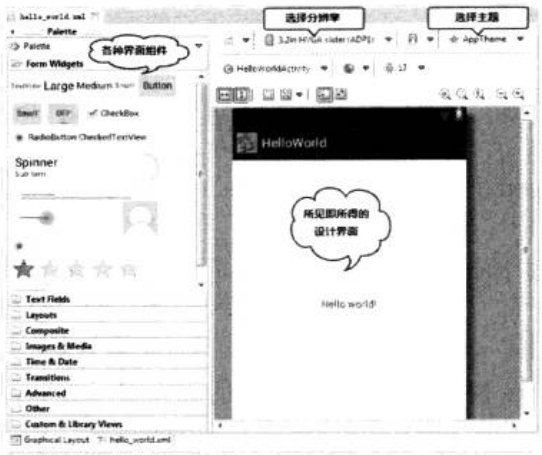


图 1.31 ADT 提供的界面设计工具

程序清单：codes\01\1.4\HelloWorld\res\layout\hello_world.xml

```
<RelativeLayout
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools">
      <TextView
            android:id="@+id/show"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:layout_alignParentTop="true"
            android:text="@string/hello_world" />
      <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_below="@id/show"
            android:text="单击我"
            android:onClick="clickHandler"/>
</RelativeLayout>
```

上面 XML 文档的根元素是 RelativeLayout，它代表了一个相对布局，在该界面布局里包含如下两个 UI 控件。

- TextView：代表一个文本框。
- Button：代表一个普通按钮。

在 Android 用户界面设计中，各种界面布局元素将会在后面进行详细介绍，不同 UI 组件也会在后面进行详细介绍。此处只想说明 UI 组件上的几个通用属性。

- android:id：该属性指定了该控件的唯一标识，在 Java 程序中可通过 findViewById("id")来获取指定的 Android 界面组件。
- android:layout_width：指定该界面组件的宽度。如果该属性值为 match_parent，则说明该组件与其父容器具有相同的宽度；如果该属性值为 wrap_content，则说明该组件的宽度取决于它的内容——能包裹它的内容即可。
- android:layout_height：指定该界面组件的高度。如果该属性值为 match_parent，则说明该组件与其父容器具有相同的高度；如果该属性值为 wrap_content，则说明

该组件的高度取决于它的内容——能包裹它的内容即可。

可能有读者感到奇怪，Android 怎么采用 XML 文件来定义用户界面呢？或者有过 Swing 编程经验的读者感到不习惯：怎么不是在 Java 代码里定义用户界面，而是在 XML 文档里定义用户界面呢？

其实大家要接受 Android 的这种优秀的设计。Android 把用户界面放在 XML 文档中定义，这样就可以让 XML 文档专门负责用户 UI 设置，而 Java 程序则专门负责业务实现，这样可以降低程序的耦合性。对于不习惯这种方式的读者来说，其实可以近似地把 hello_world.xml 文件当成一个 HTML 页面——它们都通过标记语言来定义用户界面。区别在于 HTML 页面使用 HTML 标签，而 hello_world.xml 文件则使用 Android 标签。

8 Android 项目的 src 目录是 Android 项目的源代码，该目录的 org\crazyit\helloworld 目录下有一个 HelloWorldActivity.java 文件，它就是 Android 项目的 Java 文件。打开该文件，将该文件编辑为如下形式。

程序清单：codes\01\1.4\HelloWorld\src\org\crazyit\helloworld\HelloWorldActivity.java

```
public class HelloWorldActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 使用 hello_world.xml 文件定义的界面布局
        setContentView(R.layout.hello_world);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        getMenuInflater().inflate(R.menu.hello_world, menu);
        return true;
    }
    public void clickHandler(View source)
    {
        // 获取 UI 界面中 ID 为 R.id.show 的文本框
        TextView tv = (TextView) findViewById(R.id.show);
        // 改变文本框的文本内容
        tv.setText("Hello Android-" + new java.util.Date());
    }
}
```

对于一个有不错的 Java 基础的读者来说，上面这个程序十分简单，它只做了如下两件事情。

（1）设置该 Activity 使用 hello_world.xml 文件定义的界面布局作为用户界面。

（2）定义了一个 clickHandler()方法作为按钮的事件处理方法——在处理方法中改变 ID 为 R.id.show 的文本框的内容。

至此，这个 HelloWorld 级的 Android 应用已经开发完成了。



提示：

前面介绍 hello_world.xml 文件时说过，读者可以把该文件当成一份 HTML 代码，Java 程序中通过 findViewById()方法即可获取指定 ID 的界面控件，实际上读者完全可以把 findViewById()类比成 JavaScript 代码中的 getElementById()。

1.4.2 通过 ADT 运行 Android 应用

通过 Eclipse 的 ADT 插件来运行 Android 应用非常简单，只要如下两步即可。

① 通过 Android 提供的 AVD 管理器或直接使用 emulator 命令运行指定的 AVD 设备。如果打算用真机作为运行、调试环境，使用 USB 线连接手机，并打开手机的调试模式。

② 选中需要运行的 Android 项目，单击鼠标右键，然后在弹出的快捷菜单中单击“Run As → Android Application”菜单项即可，如图 1.32 所示。

单击图 1.32 所示的菜单项之后，Eclipse 的 ADT 插件就会自动将 Android 项目安装到正在运行的模拟器或连接到电脑的真机上。此时打开“手机”，进入程序列表，即可看到刚刚开发的 HelloWorld 程序，如图 1.33 所示。

单击图 1.33 所示界面中的“第一个 Android 应用”程序项，即可在“手机”上模拟运行刚刚开发的 HelloWorld 程序，运行效果如图 1.34 所示。

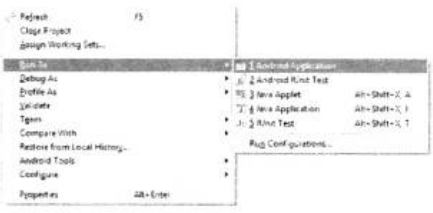


图 1.32 安装成功的 Android 应用



图 1.33 安装成功的 Android 应用



图 1.34 成功运行 Android 应用

1.5 Android 应用结构分析

使用 Eclipse 开发 Android 应用简单、方便，除了创建 Android 项目，开发者只要做两件事情：使用 main.xml 文件定义用户界面；打开 Java 源代码编写业务实现。但对于一个喜欢“穷根究底”的学习者来说，这种开发方式不免让他迷惑：

- findViewById(R.id.show);代码中的 R.id.show 是什么？从哪里来？
- 图 1.34 中程序的图标从哪里来？
- 为何 setContentView(R.layout.hello_world);代码设置使用 hello_world.xml 文件定义的界面布局？

……

实际上，喜欢“穷根究底”的学习者才是真正好的学习者。借用拿破仑的一句话：不想当将军的士兵不是好士兵。类似地也可以说：不喜欢探究原理、机制的程序员不是好程序员。

每次看到那些局限于特定 IDE 工具、只能在特定 IDE 里做事，而且自我感觉良好的学习者、工作者，笔者忍不住十分遗憾：程序员，又少了一个。

下面将带领大家“徒手”开发一个 Android 项目，这样所有的事情都是由开发者自己完成的——自然对 Android 开发的每个细节更加熟知，以后用 IDE 工具开发才能做到不仅知其

然，也知其所以然。

1.5.1 创建一个 Android 应用

前面介绍 android 命令时已经提到，该命令有一个 create project 子命令，该子命令可用于“手动”创建一个 Android 应用，在命令行窗口输入如下命令：

```
android create project -n HelloWorld -t 8 -p HelloWorld
-k org.crazyit.helloworld -a HelloWorld
```



提示：

上面的命令中，-n 选项指定创建项目的名称；-t 选项指定项目针对的 Android 平台；-p 选项指定该项目的保存路径；-k 选项指定该项目的包名；-a 选项指定 Activity 的名称。

运行上面的命令可以在当前目录的 HelloWorld 子目录下创建一个 Android 项目。进入该项目所在的目录，可以看到如下两个必要的文件夹：

```
HelloWorld
├─libs
├─res
│  ├─drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi
│  ├─layout
│  └─values
└─src（存放 Java 源文件）
   └─org
      └─crazyit
         └─helloworld
└─AndroidManifest.xml
```

上面的文件结构中 res 目录、src 目录、AndroidManifest.xml 文件是 Android 项目必需的。其他目录、其他文件都是可选的。

- res 目录存放 Android 项目的各种资源文件，比如 layout 存放界面布局文件，values 目录下则存放各种 XML 格式的资源文件，例如字符串资源文件：strings.xml；颜色资源文件：colors.xml；尺寸资源文件：dimens.xml。drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi 这 4 个子目录则分别用于存放低分辨率、中分辨率、高分辨率、超高分辨率的 4 种图片文件。
- src 目录只是一个普通的、保存 Java 源文件的目录。
- AndroidManifest.xml 文件是 Android 项目的系统清单文件，它用于控制 Android 应用的名称、图标、访问权限等整体属性。除此之外 Andriod 应用的 Activity、Service、ContentProvider、BroadcastRecevier 这 4 大组件都需要在该文件中配置。

除此之外，还可以在 HelloWorld 目录下看到一个 build.xml 文件，这是 Android 为该项目提供的一个 Ant 生成文件。通过该生成文件，开发者可以通过 Ant 来生成、安装 Android 项目。

与前面介绍的 Android 开发相似的是，此处的开发同样是先编辑 XML 格式的界面布局

文件，再编辑相应的 Java 文件。编写文件的内容与前一个示例并没有太大的区别，此处不再赘述。

编辑完成后启动命令行窗口，并转入 HelloWorld 目录下，执行 ant 命令将可以看到如图 1.35 所示的输出。

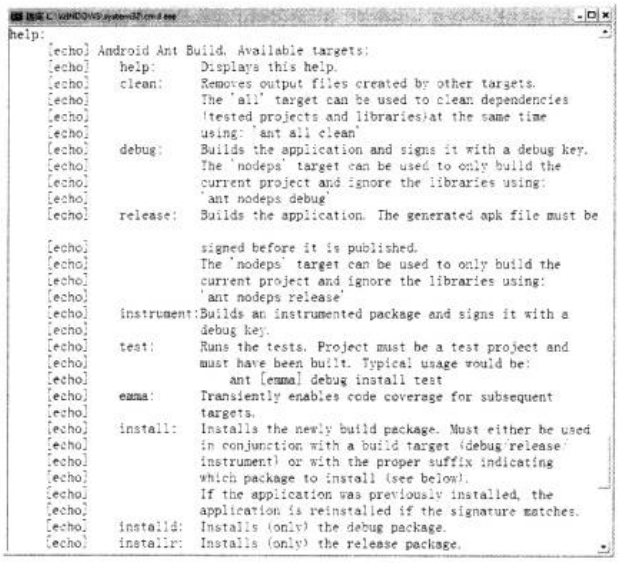


图 1.35 Android 项目的生成文件

从图 1.35 可以看出，Android 项目提供的 build.xml 文件包含如下常用的生成 target。

- clean：清除项目生成的内容——也就是恢复原来的样子。
- debug：打包一个调试用的 Android 应用的 APK 包，使用 debug key 进行签名。
- release：打包一个发布用的 Android 应用的 APK 包。
- test：运行测试。要求该项目必须是一个测试项目。
- install：将生成的调试用的 APK 包安装到模拟器上。
- uninstall：从模拟器上卸载该应用程序。



提示：

Ant 是一个非常简洁、易用的项目生成工具。对于绝大部分 Java 开发者来说，使用 Ant 应该是一项最基本的技能。考虑到有些读者对 Ant 用法不熟，此处简略介绍 Ant 的一些安装和使用方法。

① 登录 http://ant.apache.org/bindownload.cgi 站点下载 Ant 最新版，笔者使用的是 1.8.2，建议下载该版本。Windows 平台下载*.zip 压缩包，而 Linux 平台则下载.gz 压缩包。

② 将下载到的压缩文件解压缩到任意路径，例如，笔者解压缩到 D:\根路径下。

③ Ant 的运行需要如下两个环境变量：① JAVA_HOME：该环境变量应指向 JDK 的安装路径。如果已经成功安装了 Tomcat，则该环境变量应该已经是正确的。② ANT_ HOME：该环境变量应指向 Ant 的安装路径。Ant 的安装路径就是前面释放 Ant 压缩文件的路径。Ant 安装路径下应该包含 bin、docs、etc 和 lib 四个文件夹。

Ant 工具的关键命令就是%ANT_HOME%/bin 路径下的 ant.bat 命令，如果读者希望操作系统可以识别该命令，还应该将%ANT_HOME%/bin 路径添加到操作系统的 PATH 环境变量之中。

经过上面 4 个步骤，即可在命令行窗口使用 ant.bat 命令了，它就是 Ant 工具。至于 Ant 更详细的用法介绍，请参阅疯狂 Java 体系的《轻量级 Java EE 企业应用实战》。

先执行 ant release 命令来发布该项目，发布完成后看到 HelloWorld 目录下出现了两个子目录。

- bin：该目录用于存放生成的目标文件，如 Java 的二进制文件、资源打包文件（.ap_后缀）、Dalvik 虚拟机的可执行性文件（.dex 后缀）等。
- gen：该目录用于保存自动生成的、位于 Android 项目包下的 R.java 文件。

前面我们编写 Android 程序代码时多次使用了 R.layout.main、R.id.show、R.id.ok……现在读者应该明白这里 R 是什么了，原来它是 Android 项目自动生成的一个 Java 类。接下来将会详细介绍 R.java 文件。

1.5.2 自动生成的 R.java

打开 gen\org\crazyit\helloworld 目录下的 R.java 文件，看到如下代码。

程序清单：codes\01\1.5\HelloWorld\gen\org\crazyit\helloworld\R.java

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
package org.crazyit.helloworld;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int ok=0x7f050001;
        public static final int show=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

通过 R.java 类中的注释可以看出，R.java 文件是由 aapt 工具根据应用中的资源文件来自动生成的，因此可以把 R.java 理解成 Android 应用的资源字典。

aapt 生成 R.java 文件的规则主要是如下两条。

- 每类资源对应 R 类的一个内部类。比如所有界面布局资源对应于 layout 内部类；所

有字符串资源对应于 string 内部类；所有标识符资源对应于 id 内部类。

- 每个具体的资源项对应于内部类的一个 public static final int 类型的 Field。例如前面在界面布局文件中用到了 ok、show 两个标识符，因此 R.id 类里就包含了这两个 Field；由于 drawable-xxxx 文件夹里包含了 icon.png 图片，因此 R.drawable 类里包含了 icon Field。

随着我们不断地向 Android 项目中添加资源，R.java 文件的内容也会越来越多。后面还会详细介绍 Android 资源访问的相关内容，因此后面还会进一步说明不同资源在 R.java 文件中的表现形式。

1.5.3 res 目录说明

Android 应用的 res 目录是一个特殊的项目，该项目里存放了 Android 应用所用的全部资源，包括图片资源、字符串资源、颜色资源、尺寸资源等——后面还会进一步介绍 Android 应用中资源的用法，不过此处先对 res 目录的资源进行简单的归纳。

Android 按照约定，将不同的资源放在不同的文件夹内，这样可以方便地让 AAPT 工具来扫描这些资源，并为它们生成对应的资源清单类：R.java。

以/res/value/strings.xml 文件来说，该文件的内容十分简单，它只是定义了一条一条的字符串常量，如以下代码所示。

程序清单：codes\01\1.5\HelloWorld\res\values\strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">HelloWorld</string>
</resources>
```

上面的资源文件中定义了一个字符串常量，常量的值为 HelloWorld，该字符串常量的名称为 app_name。一旦定义了这份资源文件之后，Android 项目允许分别在 Java 代码、XML 代码中使用这份资源文件中的字符串资源。

1. 在 Java 代码中使用资源

为了在 Java 代码中使用资源，AAPT 会为 Android 项目自动生成一份 R.java 文件，R 类里为每份资源分别定义一个内部类，其中每个资源项对应于内部类里一个 int 类型的 Field。例如上面的字符串资源文件对应于 R.java 里的如下内容：

```
// 对应于一份资源
public static final class string {
    // 对应于一个资源项
    public static final int app_name=0x7f040000;
}
```

借助于 AAPT 自动生成 R 类的帮助，Java 代码中可通过 R.string.app_name 来引用到"HelloWorld"字符串常量。

2. 在 XML 文件中使用资源

在 XML 文件中使用资源更加简单，只要按如下格式来访问即可：

```
@<资源对应的内部类的类名>/<资源项的名称>
```

例如我们要访问上面的字符串资源中定义的"HelloWorld"字符串常量，则使用如下形式

来引用即可：

```
@string/app_name
```

但有一种情况例外，当我们在 XML 文件中使用标识符时——这些标识符无须使用专门的资源进行定义，直接在 XML 文档中按如下格式分配标识符即可：

```
@+id/<标识符代号>
```

例如使用如下代码为一个组件分配标识符：

```
android:id="@+id/ok"
```

上面的代码为该组件分配了一个标识符，接下来就可以在程序中引用该组件了。

如果希望在 Java 代码中获取该组件，通过调用 Activity 的 findViewById()方法即可实现。

如果希望在 XML 文件中获取该组件，则可通过资源引用的方式来引用它，语法如下：

```
@id/<标识符代号>
```

1.5.4 Android 应用的清单文件：AndroidManifest.xml

AndroidManifest.xml 清单文件是每个 Android 项目所必需的，它是整个 Android 应用的全局描述文件。AndroidManifest.xml 清单文件说明了该应用的名称、所使用的图标以及包含的组件等。

AndroidManifest.xml 清单文件通常可以包含如下信息：

- 应用程序的包名，该包名将会作为该应用的唯一标识。
- 应用程序所包含的组件，如 Activity、Service、BroadcastReceiver 和 Content Provider 等。
- 应用程序兼容的最低版本。
- 应用程序使用系统所需的权限声明。
- 其他程序访问该程序所需的权限声明。

不管是 Eclipse 的 ADT 工具还是 android.bat 命令，它们所创建的 Android 项目都有一个 AndroidManifest.xml 文件。但随着不断地进行开发，可能需要对 AndroidManifest.xml 清单文件进行适当的修改。

下面是一份简单的 AndroidManifest.xml 清单文件。

程序清单：codes\01\1.5\HelloWorld\AndroidManifest.xml

```
<?xml version="1.0" encoding="GBK"?>
<!-- 指定该 Android 应用的包名，
     该包名可用于唯一地表示该应用 -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.crazyit.helloworld"
    android:versionCode="1"
    android:versionName="1.0">
    <!-- 指定 Android 应用标签、图标 -->
    <application android:label="@string/app_name"
        android:icon="@drawable/ic_launcher">
        <!-- 定义 Android 应用的一个组件：Activity
            该 Activity 的类为 HelloWorld，并指定该 Activity 的标签-->
        <activity android:name="HelloWorld"
                android:label="@string/app_name">
```

```
                <intent-filter>
                    <!-- 指定该Activity是程序的入口 -->
                    <action android:name="android.intent.action.MAIN" />
                    <!-- 指定加载该应用时运行该Activity -->
                    <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
            </activity>
        </application>
    </manifest>
```

上面这份 AndroidManifest.xml 清单文件中的注释已经大致说明了各元素的作用，故不再详细说明每个元素。上面的清单文件中有两处用到了资源：

- android:label="@string/app_name"，这说明该应用的标签（Label）为/res/value 目录下 strings.xml 文件中名为 app_name 的字符串值。
- android:icon="@drawable/ic_launcher "，这说明该应用的图标为/res/drawable-l/m /hdpi 目录下主文件名为 icon 的图片。

1.5.5 应用程序权限说明

一个 Android 应用可能需要权限才能调用 Android 系统的功能；一个 Android 应用也可能被其他应用调用，因此它也需要声明调用自身所需要的权限。

1. 声明运行该应用本身所需要的权限

通过为<manifest.../>元素添加<uses-permission.../>子元素即可为程序本身声明权限。

例如在<manifest.../>元素里添加如下代码：

```
<!-- 声明该应用本身需要打电话的权限 -->
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

2. 声明调用该应用所需的权限

通过为应用的各组件元素，如<activity.../>元素添加<uses-permission.../>子元素即可声明调用该程序所需的权限。

例如在<activity.../>元素里添加如下代码：

```
<!-- 声明该应用本身需要发送短信的权限 -->
<uses-permission android:name="android.permission.SEND_SMS"/>
```

通过上面的介绍可以看出，<uses-permission.../>元素的用法倒不难，但到底有多少权限呢？实际上 Android 提供了大量的权限，这些权限都位于 Manifest.permission 类中。一般来说有如表 1.1 所示的常用权限。

表 1.1 Android 系统的常用权限

权　限	说　明
ACCESS_NETWORK_STATE	允许应用程序获取网络状态信息的权限
ACCESS_WIFI_STATE	允许应用程序获取 Wi-Fi 网络状态信息的权限
BATTERY_STATS	允许应用程序获取电池状态信息的权限
BLUETOOTH	允许应用程序连接匹配的蓝牙设备的权限
BLUETOOTH_ADMIN	允许应用程序发现匹配的蓝牙设备的权限

续表

权　限	说　明
BROADCAST_SMS	允许应用程序广播收到短信提醒的权限
CALL_PHONE	允许应用程序拨打电话的权限
CAMERA	允许应用程序使用照相机的权限
CHANGE_NETWORK_STATE	允许应用程序改变网络连接状态的权限
CHANGE_WIFI_STATE	允许应用程序改变 Wi-Fi 网络连接状态的权限
DELETE_CACHE_FILES	允许应用程序删除缓存文件的权限
DELETE_PACKAGES	允许应用程序删除安装包的权限
FLASHLIGHT	允许应用程序访问闪光灯的权限
INTERNET	允许应用程序打开网络 Socket 的权限
MODIFY_AUDIO_SETTINGS	允许应用程序修改全局声音设置的权限
PROCESS_OUTGOING_CALLS	允许应用程序监听、控制、取消呼出电话的权限
READ_CONTACTS	允许应用程序读取用户的联系人数据的权限
READ_HISTORY_BOOKMARKS	允许应用程序读取历史书签的权限
READ_OWNER_DATA	允许应用程序读取用户数据的权限
READ_PHONE_STATE	允许应用程序读取电话状态的权限
READ_PHONE_SMS	允许应用程序读取短信的权限
REBOOT	允许应用程序重启系统的权限
RECEIVE_MMS	允许应用程序接收、监控、处理彩信的权限
RECEIVE_SMS	允许应用程序接收、监控、处理短信的权限
RECORD_AUDIO	允许应用程序录音的权限
SEND_SMS	允许应用程序发送短信的权限
SET_ORIENTATION	允许应用程序旋转屏幕的权限
SET_TIME	允许应用程序设置时间的权限
SET_TIME_ZONE	允许应用程序设置时区的权限
SET_WALLPAPER	允许应用程序设置桌面壁纸的权限
VIBRATE	允许应用程序控制振动器的权限
WRITE_CONTACTS	允许应用程序写入用户联系人的权限
WRITE_HISTORY_BOOKMARKS	允许应用程序写历史书签的权限
WRITE_OWNER_DATA	允许应用程序写用户数据的权限
WRITE_SMS	允许应用程序写短信的权限

1.6　Android 应用的基本组件介绍

Android 应用通常由一个或多个基本组件组成，前面我们看到 Android 应用中最常用的组件就是 Activity。事实上 Android 应用还可能包括 Service、BroadcastReceiver、ContentProvider 等组件。本节先让读者对这些组件建立一个大致的认识，后面的章节还会对这些组件做更详细的介绍。

1.6.1　Activity 和 View

Activity 是 Android 应用中负责与用户交互的组件——大致上可以把它想象成 Swing 编程

中的 JFrame 控件。不过它与 JFrame 的区别在于：JFrame 本身可以设置布局管理器，不断地向 JFrame 中添加组件，但 Activity 只能通过 setContentView(View)来显示指定组件。

View 组件是所有 UI 控件、容器控件的基类，View 组件就是 Android 应用中用户实实在在看到的部分。但 View 组件需要放到容器组件中，或者使用 Activity 将它显示出来。如果需要通过某个 Activity 把指定 View 显示出来，调用 Activity 的 setContentView()方法即可。

setContentView()方法可接受一个 View 对象作为参数，例如如下代码：

```
// 创建一个线性布局管理器
LinearLayout layout = new LinearLayout(this);
// 设置该 Activity 显示 layout
setContentView(layout);
```

上面的程序通过代码创建了一个 LinearLayout 对象（它是 ViewGroup 的子类，ViewGroup 又是 View 的子类），接着调用 Activity 的 setContentView(layout)把这个布局管理器显示出来。

setContentView()方法也可接受一个布局管理资源的 ID 作为参数，例如如下代码：

```
// 设置该 Activity 显示 main.xml 文件定义的 View
setContentView(R.layout.main);
```

从这个角度来看，大致上可以把 Activity 理解成 Swing 中的 JFrame 组件。当然，Activity 可以完成的功能比 JFrame 更多，此处只是简单地类比一下。



提示： 实际上 Activity 是 Window 的容器，Activity 包含一个 getWindow()方法，该方法返回该 Activity 所包含的窗口。对于 Activity 而言，开发者一般不需要关心 Window 对象。如果应用程序不调用 Activity 的 setContentView()来设置该窗口显示的内容，那么该程序将显示一个空窗口。

Activity 为 Android 应用提供了可视化用户界面，如果该 Android 应用需要多个用户界面，那么这个 Android 应用将会包含多个 Activity，多个 Activity 组成 Activity 栈，当前活动的 Activity 位于栈顶。

Activity 包含了一个 setTheme(int resid)方法来设置其窗口的风格，例如我们希望窗口不显示 ActionBar、以对话框形式显示窗口，都可通过该方法来实现。

1.6.2 Service

Service 与 Activity 的地位是并列的，它也代表一个单独的 Android 组件。Service 与 Activity 的区别在于：Service 通常位于后台运行，它一般不需要与用户交互，因此 Service 组件没有图形用户界面。

与 Activity 组件需要继承 Activity 基类相似，Service 组件需要继承 Service 基类。一个 Service 组件被运行起来之后，它将拥有自己独立的生命周期，Service 组件通常用于为其他组件提供后台服务或监控其他组件的运行状态。

1.6.3 BroadcastReceiver

BroadcastReceiver 是 Android 应用中另一个重要的组件，顾名思义，BroadcastReceiver 代表广播消息接收器。从代码实现角度来看，BroadcastReceiver 非常类似于事件编程中的监

听器。与普通事件监听器不同的是：普通事件监听器监听的事件源是程序中的对象；而BroadcastReceiver监听的事件源是Android应用中的其他组件。

使用BroadcastReceiver组件接收广播消息比较简单，开发者只要实现自己的BroadcastReceiver子类，并重写onReceive(Context context, Intent intent)方法即可。当其他组件通过sendBroadcast()、sendStickyBroadcast()或sendOrderedBroadcast()方法发送广播消息时，如该BroadcastReceiver也对该消息“感兴趣”（通过IntentFilter配置），BroadcastReceiver的onReceive(Context context, Intent intent)方法将会被触发。

开发者实现了自己的BroadcastReceiver之后，通常有两种方式来注册这个系统级的“事件监听器”。

- 在Java代码中通过Context.registReceiver()方法注册BroadcastReceiver。
- 在AndroidManifest.xml文件中使用<receiver.../>元素完成注册。

读者此处只要对BroadcastReceiver有一个大致的印象即可，本书后面的章节还会详细介绍如何开发、使用BroadcastReceiver组件。

1.6.4 ContentProvider

对于Android应用而言，它们必须相互独立，各自运行在自己的Dalvik虚拟机实例中，如果这些Android应用之间需要实现实时的数据交换。例如我们开发了一个发送短信的程序，当发送短信时需要从联系人管理应用中读取指定联系人的数据——这就需要多个应用程序之间进行数据交换。

Android系统为这种跨应用的数据交换提供了一个标准：ContentProvider。当用户实现自己的ContentProvider时，需要实现如下抽象方法。

- insert(Uri, ContentValues)：向ContentProvider插入数据。
- delete(Uri, ContentValues)：删除ContentProvider中指定数据。
- update(Uri, ContentValues, String, String[])：更新ContentProvider中指定数据。
- query(Uri, String[], String, String[],String)：从ContentProvider查询数据。

通常与ContentProvider结合使用的是ContentResolver，一个应用程序使用ContentProvider暴露自己的数据，而另一个应用程序则通过ContentResolver来访问数据。

1.6.5 Intent和IntentFilter

严格地说，Intent并不是Android应用的组件，但它对于Android应用的作用非常大——它是Android应用内不同组件之间通信的载体。当Android运行时需要连接不同的组件时，通常就需要借助于Intent来实现。Intent可以启动应用中另一个Activity，也可以启动一个Service组件，还可以发送一条广播消息来触发系统中的BroadcastReceiver。也就是说，Activity、Service、BroadcastReceiver三种组件之间的通信都以Intent作为载体，只是不同组件使用Intent的机制略有区别而已。

- 当需要启动一个Activity时，可调用Context的startActivity (Intent intent)或startActivityForResult(Intent intent, int requestCode)方法，这两个方法中的Intent参数封装了需要启动的目标Activity的信息。
- 当需要启动一个Service时，可调用Context的startService(Intent intent)方法或bindService(Intent service, ServiceConnection conn, int flags)方法，这两个方法中

的 Intent 参数封装了需要启动的目标 Service 的信息。

- 当需要触发一个 BroadcastReceiver 时，可调用 Context 的 sendBroadcast(Intent intent)、sendStickyBroadcast(Intent intent)或 sendOrderedBroadcast(Intent intent, String receiverPermission)方法来发送广播消息，这三个方法中的 Intent 参数封装了需要触发的目标 BroadcastReceiver 的信息。

通过上面的介绍不难看出，Intent 封装了当前组件需要启动或触发的目标组件的信息，因此有些书上把 Intent 翻译为“意图”。实际上 Intent 对象里封装了大量关于目标组件的信息，本书后面还会更详细地介绍 Intent 所封装的数据，此处不再深入讲解。

当一个组件通过 Intent 表示了启动或触发另一个组件的“意图”之后，这个意图可分为两类。

- 显式 Intent：显式 Intent 明确指定需要启动或者触发的组件的类名。
- 隐式 Intent：隐式 Intent 只是指定需要启动或者触发的组件应满足怎样的条件。

对于显式 Intent 而言，Android 系统无须对该 Intent 做任何解析，系统直接找到指定的目标组件，启动或触发它即可。

对于隐式 Intent 而言，Android 系统需要对该 Intent 进行解析，解析出它的条件，然后再去系统中查找与之匹配的目标组件。如果找到符合条件的组件，就启动或触发它们。

那么 Android 系统如何判断被调用组件是否符合隐式 Intent 呢？这就需要靠 IntentFilter 来实现了，被调用组件可通过 IntentFilter 来声明自己所满足的条件——也就是声明自己到底能处理哪些隐式 Intent。关于 Intent 和 IntentFilter 本书后面还会有进一步阐述，此处不再深入讲解。

1.7 签名 Android 应用程序

前面已经介绍过：Android 项目以它的包名作为唯一标识。如果在同一台手机上安装两个包名相同的应用，后面安装的应用就可以覆盖前面安装的应用。为了避免这种情况发生，Android 要求对作为产品发布的应用进行签名。

签名主要有如下两个作用。

- 确定发布者的身份。由于应用开发者可以通过使用相同包名来替换已经安装的程序，因此使用签名可以避免发生这种情况。
- 确保应用的完整性。签名会对应用包中的每个文件进行处理，从而确保程序包中的文件不会被替换。

通过以上介绍不难看出，Android 应用签名的作用类似于现实生活中的签名。当开发者对 Android 应用签名时，相当于告诉外界：该应用程序是由“我”开发的，“我”会对该应用负责——因为有签名（签名有密钥），别人无法冒名顶替“我”；与此同时，“我”也无法冒名顶替别人。



提示： 在应用的开发、调试阶段，Eclipse 的 ADT 插件或 Ant 工具会自动生成调试证书对 Android 应用签名，因此部署、调试前面两个示例并没有经过签名。需要指出的是，如果要正式发布一个 Android 应用，必须使用合适的数字证书来给应用程序签名，不能使用 ADT 插件或 Ant 工具生成的调试证书来发布。

1.7.1　在 Eclipse 中对 Android 应用签名

大部分时候，开发者会直接在 Eclipse 中对 Android 应用签名，在 Eclipse 中对 Android 应用签名的步骤如下。

① 右击 Android 项目，单击如图 1.36 所示“Android Tools→Export Signed Application Package...”菜单项，Eclipse 弹出如图 1.37 所示窗口。

② 如果系统中还没有数字证书，可以在图 1.37 所示窗口中选中“Create new keystore”单选按钮，并按图 1.37 所示格式填写数字证书的存储路径和密码。

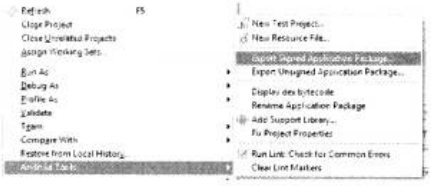


图 1.36　导出签名的 Android 应用程序



图 1.37　填写数字证书的存储路径和密码

③ 按图 1.37 所示格式填写完成后单击“Next”按钮，Eclipse 将会出现如图 1.38 所示窗口，该窗口让用户填写数字证书的详细信息。

④ 按图 1.38 所示形式为数字证书填写详细信息后，单击“Next”按钮，Eclipse 打开如图 1.39 所示窗口，该窗口用于指定生成签名后的 APK 安装包的存储路径。

⑤ 单击图 1.39 所示窗口中的“Finish”按钮，签名完成。Eclipse 将会在指定路径下生成一个签名后的 APK 安装包。

上面步骤的第（4）步用于制作新的数字证书，一旦数字证书制作完成，以后就可以直接使用该数字证书签名了。

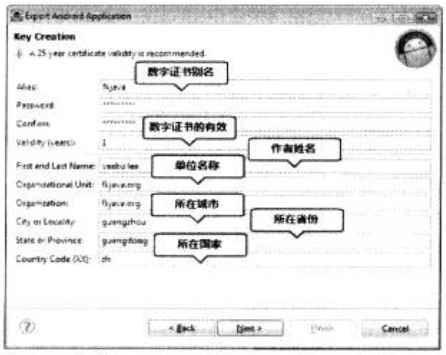


图 1.38　填写数字证书的详细信息



图 1.39　指定签名后的 APK 安装包的存储路径

利用已有的数字证书进行签名，请按如下步骤进行。

① 在图 1.37 所示窗口中选中“Use existing keystore”，并使用前面创建的数字证书，如图 1.40 所示。

② 在图 1.40 所示窗口中选中前面创建的 keystore，并输入创建 keystore 时指定的密码，然后单击“Next”按钮，Eclipse 显示如图 1.41 所示的窗口。

③ 在图 1.41 中选择签名创建的别名为 fkjava 的 key，并输入创建该 key 时指定的密码。然后单击该窗口中“Next”按钮，Eclipse 将显示图 1.39 所示窗口，用户通过该窗口指定签名

后 APK 包的存储路径，最后单击“Finish”按钮即可生成签名后 APK 包。



图 1.40 使用已有的数字签名

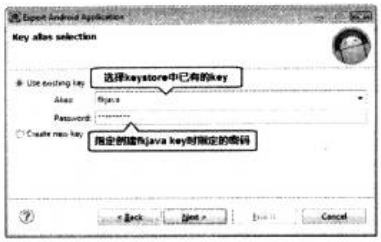


图 1.41 选择已有的 key

1.7.2 使用命令对 APK 包签名

如果不想借助于 Eclipse 提供的方式对 Android 应用程序签名，或在某些场合下，需要对一个“未签名”的 APK 包进行签名，则可通过“命令”来对 Android 应用进行手动签名。

使用命令对 Android 应用签名的步骤如下。

① 创建 keystore 库。JDK 的安装目录下的 bin 子目录下提供了 keytool.exe 工具来生成数字证书。在命令行窗口输入如下命令：

```
keytool -genkeypair -alias crazyit.keystore -keyalg RSA -validity 400
 -keystore crazyit.keystore
```

上面命令中各选项说明如下。

- -genkeypair：指定生成数字证书。
- -alias：指定生成数字证书的别名。
- -keyalg：指定生成数字证书的算法。使用 RSA 算法。
- -validity：指定生成的数字证书的有效期。
- - keystore：指定所生成的数字证书的存储路径。

输入上面命令后按回车键，接下来将会以交互式方式让用户输入数字证书 keystore 的密码、作者、公司等详细信息，如图 1.42 所示。



图 1.42 生成数字证书



提示： 第（1）步的作用是生成属于你们公司、你的数字证书，这个步骤只要做一次即可。一旦数字证书创建成功之后，只要在该证书有效期内，可以一直重复使用该证书。

② 生成未签名的 APK 安装包。在 Eclipse 中右击 Android 项目，单击如图 1.43 所示“Android Tools→Export Unsigned Application Package...”菜单项，Eclipse 弹出一个保存文件的对话框，当用户选择存储文件后单击“Finish”按钮即可生成一个未签名的 APK 安装包。

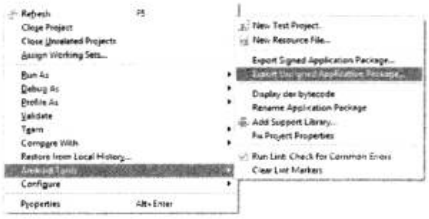


图 1.43　导出未签名的 Android 应用程序



提示：

第（2）步作用是生成一个未签名的 APK 安装包，如果本来已有了这个未签名的安装包，或者该安装包是你们委托第三方公司开发的、第三方公司负责提供该未签名的安装包，那么这个步骤是可以省略的。

③ 使用 jarsigner 命令对未签名的 APK 安装包进行签名。JDK 的安装目录下的 bin 子目录下提供了 jarsigner.exe 工具进行签名。在命令行窗口输入如下命令：

```
jarsigner -verbose -keystore crazyit.keystore -signedjar
 HelloWorld_crazyit.apk HelloWorld.apk crazyit.keystore
```

上面的命令中各选项说明如下。

- **-verbose**：指定生成详细输出。
- **-keystore**：指定数字证书的存储路径。
- **-signedjar**：该选项的三个参数分别为签名后的 APK 包、未签名的 APK 包、数字证书的别名。

输入上面命令后按回车键，接下来将会以交互式方式让用户输入数字证书 keystore 的密码，如图 1.44 所示。



图 1.44　执行数字签名

④ 使用 zipalign.exe 工具优化 APK 安装包。zipalign.exe 是 Android 自带的一个档案整理工具，它可用于优化 APK 安装包，从而提升 Android 应用与系统之间的交互效率，提升应用程序的运行速度。在命令行窗口输入如下命令：

```
zipalign -f -v 4 HelloWorld_crazyit.apk
```

```
HelloWorld_crazyit_zip.apk
```

上面的命令中各选项说明如下。

- -f：指定强制覆盖已有的文件。
- -v：指定生成详细输出。
- 4：指定档案整理所基于的字节数，通常指定为 4，也就是基于 32 位进行整理。
- HelloWorld_crazyit.apk 和 HelloWorld_crazyit_zip.apk 分别指定整理前的 APK 和整理后生成的 APK。

运行上面命令，将会在当前目录下生成一个 HelloWorld_crazyit_zip.apk 文件，这就是签名完成且经过优化的 APK 安装包，该安装包可以对外发布了。

1.8 本章小结

本章简要介绍了 Android 应用开发的背景知识，包括 Android 是什么、它是干什么的；简要介绍了 Android 的发展历史及现状。读者本章需要掌握的重点是搭建、使用 Android 开发平台，包括下载和安装 Android SDK、下载、安装、使用 ADT 工具；除此之外，Android SDK 提供的各种小工具，如 ADB、DDMS、DX、AAPT 也是需要重点掌握的，这些内容是开发 Android 应用的基础。除此之外，本章介绍了一个 Android 的 Hello World 应用，分别介绍通过 Eclipse 工具开发和不使用工具开发两种方式；通过介绍不使用任何工具来开发 Android 应用，可以让读者对 Android 应用的程序结构更加熟悉。除此之外，本章详细介绍了 Android 应用的 AndroidManifest.xml 文件，以及如何在该文件中管理程序权限。

CHAPTER

2

第2章 Android 应用的界面编程

本章要点

- Android 的程序界面与 View 组件
- View 组件与 ViewGroup 组件
- Android 控制程序界面的三种方式
- 通过继承 View 开发自定义 View
- Android 常见的布局管理器
- 文本框组件：TextView 和 EditText
- 按钮组件：Button
- 特殊的按钮组件：RadioButton、CheckBox、ToggleButton 和 Switch
- 时间显示组件：AnalogClock 与 DigitalClock
- 图片浏览组件：ImageView
- ImageButton、ZoomButton 与 QuickContactBadge
- AdapterView 组件与 Adapter
- ListView 与 GridView 的功能与用法
- ExpandableListView 组件的功能与用法
- Spinner 与 Gallery 的功能与用法
- AutoCompleteTextView 组件的功能与用法
- AdapterViewFlipper 与 StackView 的功能和用法
- ProgressBar 进度条的功能与用法
- SeekBar 的功能与用法
- RatingBar 的功能与用法
- ViewAnimator 与 ViewSwitch 的功能与用法
- ImageSwitch、TextSwitch 与 ViewFlipper 的功能与用法
- 使用 Toast 创建简单提示
- CalendarView 的功能与用法
- DatePicker、TimerPiack 与 NumberPicker 的功能与用法
- SearchView 的功能与用法
- TabHost 的功能与用法
- ScrollView 的功能与用法
- 使用 Notification 发送全局通知
- 使用 AlertDialog 创建对话框
- 使用 AlertDialog 创建各种复杂的对话框
- 对话框风格的窗口
- 使用 PopupWindow 创建对话框
- 开发选项菜单和子菜单
- 为菜单项提供响应
- 使用 ActionBar 显示选项菜单
- 为 ActionBar 添加 ActionView
- ActionBar 结合 Fragment 实现 Tab 导航
- ActionBar 结合 Fragment 实现下拉式导航

Android 应用开发的一项内容就是用户界面的开发。不管应用实际包含的逻辑多么复杂，多么优秀，如果这个应用没有提供友好的图形用户界面，将很难吸引最终用户。相反，如果为应用程序提供友好的图形用户界面（Graphics User Interface，GUI），最终用户通过鼠标拖动、点击等动作就可以操作整个应用，这个应用程序就会受欢迎得多（实际上，Windows 之所以广为人知，其最初的吸引力就是来自于它所提供的图形用户界面）。作为一个程序设计者，必须优先考虑用户的感受，一定要让用户感到“爽”，我们的程序才会被需要、被使用，这样的程序才有价值。

Android 提供了大量功能丰富的 UI 组件，开发者只要按一定规律把这些 UI 组件组合起来——就像小朋友“搭积木”一样，把这些 UI 组件搭建在一起就可以开发出优秀的图形用户界面。为了让这些 UI 组件能响应用户的鼠标、键盘动作，Android 也提供了事件响应机制，这样保证图形界面应用可响应用户的交互操作。

通过学习本章，读者应该能开发出漂亮的图形用户界面，这些图形用户界面是 Android 应用开发的基础，也是非常重要的组成部分。

2.1 界面编程与视图（View）组件

Android 应用是运行于手机系统上的程序，这种程序给用户的第一印象就是用户界面。从市场的角度来看，所有开发者都应充分重视 Android 应用的用户界面。Android 提供了非常丰富的用户界面组件，借助于这些用户界面组件，开发者可以非常方便地进行用户界面开发，而且可以开发出非常优秀的用户界面。

➤➤2.1.1 视图组件与容器组件

Android 应用的绝大部分 UI 组件都放在 android.widget 包及其子包、android.view 包及其子包中，Android 应用的所有 UI 组件都继承了 View 类，View 组件非常类似于 Swing 编程的 JPanel，它代表一个空白的矩形区域。

View 类还有一个重要的子类：ViewGroup，但 ViewGroup 通常作为其他组件的容器使用。

Android 的所有 UI 组件都是建立在 View、ViewGroup 基础之上的，Android 采用了“组合器”设计模式来设计 View 和 ViewGroup：ViewGroup 是 View 的子类，因此 ViewGroup 也可被当成 View 使用。对于一个 Android 应用的图形用户界面来说，ViewGroup 作为容器来盛装其他组件，而 ViewGroup 里除了可以包含普通 View 组件之外，还可以再次包含 ViewGroup 组件。

图 2.1 显示了 Android 图形用户界面的组件层次图。

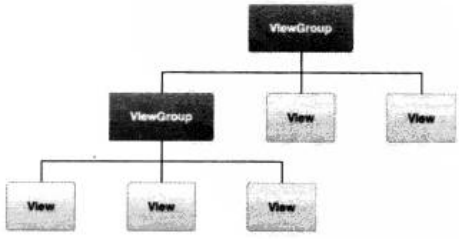


图 2.1 图形用户界面的组件层次

图 2.1 来自 Android 文档。对于每个 Android 开发者而言，Android 提供的官方文档是必看的。下面简单介绍读者应该如何查看 Android 文档——这实际上是一种学习方法。实际上，笔者常常觉得掌握学习方法比记住几个知识点更重要。

在第 1 章在线安装 Android SDK 组件时，通过图 1.3 所示窗口选择 Android 工具时应该勾选“Documentation for Android SDK”项，就会将 Android 文档安装到本地磁盘。一旦我们

将 Android 文档安装到本地磁盘，就可以在 Android SDK 安装目录找到 docs 子目录，打开 docs 子目录下的 index.html 页面，并单击该页面上方的 Develop→API Guides（开发指南）标签页，用户将看到如图 2.2 所示页面。

图 2.2 所示就是 Android 官方提供的开发指南文档，这份文档也是笔者当初学习、开发 Android 应用的重要文档。



图 2.2　Android 开发指南



提示：

如果具有良好的英文阅读能力，而且 Java 基本功扎实，学习 Android 完全可以不用购买任何图书，直接阅读这份 API Guides 也是很好的学习方法。

单击图 2.2 所示页面的 Reference 标签页，接下来所看到的就是 Android 的 API 文档，如图 2.3 所示。



图 2.3　Android 的 API 文档

图 2.3 所示的 API 文档与我们熟悉的 API 文档大致相同，最大的区别在于 Android 的 API 文档并未在类列表区直接列出所有类，只有当开发者选择指定包之后，类列表区才会列出该

包下的所有类，这给开发者带来了一些不便。建议使用 Chrome、Firefox 等浏览器查看这份 API 文档，使用 IE 查看这份 API 文档会比较慢。这份 API 文档与所有 API 文档一样，都是开发者必须时常查阅的手册。



提示： 图 2.3 所示页面中 API Guides、Reference 两个标签页的内容是 Android 文档的最重要内容，不过本书内容将会基本覆盖 API Guides 的内容。不仅如此，Resource 标签页中也包含了 Android 的一些简单的入门示例，也是初学者学习 Android 应用开发不错的资料。

前面介绍 Android 应用结构时已经指出：Android 推荐使用 XML 布局文件来定义用户界面，而不是使用 Java 代码来开发用户界面，因此所有组件都提供了两种方式来控制组件的行为：

- 在 XML 布局文件中通过 XML 属性进行控制。
- 在 Java 程序代码中通过调用方法进行控制。

实际上不管使用哪种方式，它们控制 Android 用户界面行为的本质是完全一样的。大部分时候，控制 UI 组件的 XML 属性还有对应的方法。

对于 View 类而言，它是所有 UI 组件的基类，因此它包含的 XML 属性和方法是所有组件都可使用的。表 2.1 是 View 类常用的 XML 属性、相关方法及简要说明。

表 2.1 View 类的 XML 属性、相关方法及说明

XML 属性	相关方法	说　明
android:alpha	setAlpha(float)	设置该组件的透明度
android:background	setBackgroundResource(int)	设置该组件的背景颜色
android:clickable	setClickable(boolean)	设置该组件是否可以激发单击事件
android:contentDescription	setContentDescription(CharSequence)	设置该组件的主要描述信息
android:drawingCacheQuality	setDrawingCacheQuality(int)	设置该组件所使用的绘制缓存的质量
android:fadeScrollbars	setScrollbarFadingEnabled(boolean)	当不使用该组件的滚动条时，是否淡出显示滚动条
android:fadingEdge	setVerticalFadingEdgeEnabled(boolean)	设置滚动该组件时组件边界是否使用淡出效果
android:fadingEdgeLength	getVerticalFadingEdgeLength()	设置淡出边界的长度
android:focusable	setFocusable(boolean)	设置该组件是否可以得到焦点
android:focusableInTouchMode	setFocusableInTouchMode(boolean)	设置该组件在触摸模式下是否可以得到焦点
android:id	setId(int)	设置该组件的唯一标识。Java 代码中可通过 findViewById 来获取它
android:isScrollContainer	setScrollContainer(boolean)	设置该组件是否是作为可滚动容器使用
android:keepScreenOn	setKeepScreenOn(boolean)	设置该组件是否会强制手机屏幕一直打开
android:longClickable	setLongClickable(boolean)	设置该组件是否可以响应长单击事件
android:minHeight	setMinimumHeight(int)	设置该组件的最小高度
android:minWidth	setMinimumWidth(int)	设置该组件的最小宽度

续表

XML 属性	相关方法	说　明
android:nextFocusDown	setNextFocusDownId(int)	设置焦点在该组件上，且按向下键时获得焦点的组件 ID
android:nextFocusLeft	setNextFocusLeftId(int)	设置焦点在该组件上，且按向左键时获得焦点的组件 ID
android:nextFocusRight	setNextFocusRightId(int)	设置焦点在该组件上，且按向右键时获得焦点的组件 ID
android:nextFocusUp	setNextFocusUpId(int)	设置焦点在该组件上，且按向上键时获得焦点的组件 ID
android:onClick		为该组件的单击事件绑定监听器
android:padding	setPadding(int,int,int,int)	在组件的四边设置填充区域
android:paddingBottom	setPadding(int,int,int,int)	在组件的下边设置填充区域
android:paddingLeft	setPadding(int,int,int,int)	在组件的左边设置填充区域
android:paddingRight	setPadding(int,int,int,int)	在组件的右边设置填充区域
android:paddingTop	setPadding(int,int,int,int)	在组件的上边设置填充区域
android:rotation	setRotation(float)	设置该组件旋转的角度
android:rotationX	setRotationX(float)	设置该组件绕 *X* 轴旋转的角度
android:rotationY	setRotationY(float)	设置该组件绕 *Y* 轴旋转的角度
android:saveEnabled	setSaveEnabled(boolean)	如果设置为 false，那当该组件被冻结时不会保存它的状态
android:scaleX	setScaleX(float)	设置该组件在水平方向的缩放比
android:scaleY	setScaleY(float)	设置该组件在垂直方向的缩放比
android:scrollX		该组件初始化后的水平滚动偏移
android:scrollY		该组件初始化后的垂直滚动偏移
android:scrollbarAlwaysDrawHorizontalTrack		设置该组件是否总是显示水平滚动条的轨道
android:scrollbarAlwaysDrawVerticalTrack		设置该组件是否总是显示垂直滚动条的轨道
android:scrollbarDefaultDelayBeforeFade	setScrollBarDefaultDelayBeforeFade(int)	设置滚动条在淡出隐藏之前延迟多少毫秒
android:scrollbarFadeDuration	setScrollBarFadeDuration(int)	设置滚动条淡出隐藏过程需要多少秒
android:scrollbarSize	setScrollBarSize(int)	设置垂直滚动条的宽度和水平滚动条的高度
android:scrollbarStyle	setScrollBarStyle(int)	设置滚动条的风格和位置。该属性支持如下属性值： insideOverlay insideInset outsideOverlay outsideInset
android:scrollbarThumbHorizontal		设置该组件的水平滚动条的滑块对应的 Drawable 对象

续表

XML 属性	相关方法	说　明
android:scrollbarThumbVertical		设置该组件的垂直滚动条的滑块对应的 Drawable 对象
android:scrollbarTrackHorizontal		设置该组件的水平滚动条的轨道对应的 Drawable 对象
android:scrollbarTrackVertical		设置该组件的垂直滚动条的轨道对应的 Drawable 对象
android:scrollbars		定义该组件滚动时显示几个滚动条。该属性支持如下属性值。 none：不显示滚动条 horizontal：显示水平滚动条 vertical：显示垂直滚动条
android:soundEffectsEnabled	setSoundEffectsEnabled(boolean)	设置该组件被单击时是否使用音效
android:tag		为该组件设置一个字符串类型的 tag 值。接下来可通过 View 的 getTag()获取该字符串，或通过 findViewWithTag()查找该组件
android:transformPivotX	setPivotX(float)	设置该组件旋转时旋转中心的 *X* 坐标
android:transformPivotY	setPivotY(float)	设置该组件旋转时旋转中心的 *Y* 坐标
android:translationX	setTranslationX(float)	设置该组件在 *X* 方向上的位移
android:translationY	setTranslationY(float)	设置该组件在 *Y* 方向上的位移
android:visibility	setVisibility(int)	设置该组件是否可见



提示： Drawable 是 Android 提供的一个抽象基类，它代表了“可以被绘制出来的某种东西”，Drawable 包括了大量子类，比如 BitmapDrawable 代表位图 Drawable、ColorDrawable 代表颜色 Drawable、ShapeDrawable 代表几何形状 Drawable。各种 Drawable 可用于定制 UI 组件的背景等外观。本书第 7 章会详细介绍各种 Drawable 的功能与用法。

ViewGroup 继承了 View 类，当然也可以当成普通 View 来使用，但 ViewGroup 主要还是当成容器类使用。但由于 ViewGroup 是一个抽象类，因此实际使用中通常总是使用 ViewGroup 的子类来作为容器，例如各种布局管理器。

ViewGroup 容器控制其子组件的分布依赖于 ViewGroup.LayoutParams、ViewGroup.MarginLayoutParams 两个内部类。这两个内部类中都提供了一些 XML 属性，ViewGroup 容器中的子组件可以指定这些 XML 属性。

表 2.2 显示了 ViewGroup.LayoutParams 所支持的两个 XML 属性。

表 2.2　ViewGroup 子元素支持的属性

XML 属性	说　明
android:layout_height	指定该子组件的布局高度
android:layout_width	指定该子组件的布局宽度

android:layout_height、android:layout_width 两个属性支持如下三个属性值。

- fill_parent：指定子组件的高度、宽度与父容器组件的高度、宽度相同（实际上还要减去填充的空白距离）。
- match_parent：该属性值与 fill_parent 完全相同，而且从 Android 2.2 开始就推荐使用这个属性值来代替 fill_parent。
- wrap_content：指定子组件的大小恰好能包裹它的内容即可。



提示：

虽然 Android 推荐使用 match_parent 代替 fill_parent，但由于通过 ADT 生成的 UI 组件有些地方依然使用了 fill_parent 属性值，因此本书示例程序中还有少量地方使用了 fill_parent 属性值。

读者可能对布局高度与布局宽度感到疑惑：为组件指定了高度与宽度不就够了吗？为何还要设置布局高度与布局宽度呢？这是由 Android 的布局机制决定的，Android 组件的大小不仅受它实际的宽度、高度控制，还受它的布局高度与布局宽度控制。比如设置一个组件的宽度为 30px，如果将它的布局宽度设为 match_parent，那么该组件的宽度将会被“拉宽”到占满它所在的父容器；如果将它的布局宽度设为 wrap_content，那么该组件的宽度才会是 30px。

表 2.3 显示了 ViewGroup.MarginLayoutParams 用于控制子组件周围的页边距（Margin，也就是组件四周的留白），它支持的 XML 属性如表 2.3 所示。

表 2.3 ViewGroup.MarginLayoutParams 支持的属性

XML 属性	相关方法	说 明
android:layout_marginBottom	setMargins(int,int,int,int)	指定该子组件下边的页边距
android:layout_marginLeft	setMargins(int,int,int,int)	指定该子组件左边的页边距
android:layout_marginRight	setMargins(int,int,int,int)	指定该子组件右边的页边距
android:layout_marginTop	setMargins(int,int,int,int)	指定该子组件上边的页边距

后面我们还会详细介绍 ViewGroup 各子类的用法，此处不再详述。

2.1.2 使用 XML 布局文件控制 UI 界面

Android 推荐使用 XML 布局文件来控制视图，这样不仅简单、明了，而且可以将应用的视图控制逻辑从 Java 代码中分离出来，放入 XML 文件中控制，从而更好地体现 MVC 原则。

当我们在 Android 应用的 res/layout 目录下定义一个主文件名任意的 XML 布局文件之后（R.java 会自动收录该布局资源），Java 代码可通过如下方法在 Activity 中显示该视图：

```
setContentView(R.layout.<资源文件名字>);
```

当在布局文件中添加多个 UI 组件时，都可以为该 UI 组件指定 android:id 属性，该属性的属性值代表该组件的唯一标识。接下来如果希望在 Java 代码中访问指定 UI 组件，可通过如下代码来访问它：

```
findViewById(R.id.<android.id 属性值>);
```

一旦在程序中获得指定 UI 组件之后，接下来就可以通过代码来控制各 UI 组件的外观行为了，包括为 UI 组件绑定事件监听器等。

2.1.3 在代码中控制 UI 界面

虽然 Android 推荐使用 XML 布局文件来控制 UI 界面，但如果开发者愿意，Android 允许开发者像开发 Swing 应用一样，完全抛弃 XML 布局文件，完全在 Java 代码中控制 UI 界面。如果希望在代码中控制 UI 界面，那么所有的 UI 组件都将通过 new 关键字创建出来，然后以合适的方式“搭建”在一起即可。

实例：用编程式的方式开发 UI 界面

下面将试图开发一个完全用代码控制 UI 界面的 Android 应用。由于该应用完全采用代码来控制 UI 界面，因此可以完全抛弃 XML 布局文件。下面是通过代码控制 UI 界面的代码。

程序清单：codes\02\2.1\CodeView\src\org\crazyit\ui\CodeView.java

```
public class CodeView extends Activity
{
    // 当第一次创建该 Activity 时回调该方法
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 创建一个线性布局管理器
        LinearLayout layout = new LinearLayout(this);
        // 设置该 Activity 显示 layout
        super.setContentView(layout);
        layout.setOrientation(LinearLayout.VERTICAL);
        // 创建一个 TextView
        final TextView show = new TextView(this);
        // 创建一个按钮
        Button bn = new Button(this);
        bn.setText(R.string.ok);
        bn.setLayoutParams(new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT));
        // 向 Layout 容器中添加 TextView
        layout.addView(show);
        // 向 Layout 容器中添加按钮
        layout.addView(bn);
        // 为按钮绑定一个事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                show.setText("Hello , Android , " + new java.util.Date());
            }
        });
    }
}
```

从上面的程序的粗体字代码可以看出，该程序中所用到的 UI 组件都是通过 new 关键字创建出来的，然后程序使用 LinearLayout 容器来“盛装”这些 UI 组件，这样就组成了图形用户界面。

从上面的程序代码中可以看出，无论创建哪种 UI 组件，都需要传入一个 this 参数，这是由于创建 UI 组件时传入一个 Context 参数，Context 代表访问 Android 应用环境的全局信

息的 API。让 UI 组件持有一个 Context 参数，可让这些 UI 组件通过该 Context 参数来获取 Android 应用环境的全局信息。



图 2.4　在代码中控制 UI 界面

Context 本身是一个抽象类，Android 的应用的 Activity、Service 都继承了 Context，因此 Activity、Service 都可直接作为 Context 使用。

在模拟器中运行上面的程序将可以看到如图 2.4 所示界面。

从上面的程序代码中不难看出，完全在代码中控制 UI 界面不仅不利于高层次的解耦，而且由于通过 new 关键来创建 UI 组件，需要调用方法来设置 UI 组件的行为，因此代码也显得十分臃肿；相反，如果通过 XML 文件来控制 UI 界面，开发者只要在 XML 布局文件中使用标签即可创建 UI 组件，而且只要配置简单的属性即可控制 UI 组件的行为，因此要简单得多。

虽然 Android 应用完全允许开发者像开发 Swing 应用一样在代码中控制 UI 界面，但这种方式不仅编程烦琐，而且不利于高层次的解耦，因此不推荐开发者使用这种方式。

➤➤2.1.4　使用 XML 布局文件和 Java 代码混合控制 UI 界面

前面已经提到，完全使用 Java 代码来控制 UI 界面不仅烦琐、而且不利于解耦；而完全利用 XML 布局文件来控制 UI 界面虽然方便、便捷，但难免有失灵活。因此有些时候，可能需要混合使用 XML 布局文件和代码来控制 UI 界面。

当混合使用 XML 布局文件和代码来控制 UI 界面时，习惯上把变化小、行为比较固定的组件放在 XML 布局文件中管理，而那些变化较多、行为控制比较复杂的组件则交给 Java 代码来管理。

实例：简单图片浏览器

例如下面的应用，我们先在布局文件中定义一个简单的线性布局容器，该布局文件的代码如下。

程序清单：codes\02\2.1\MixView\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个线性布局容器 -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/root"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
</LinearLayout>
```

上面的布局文件只是定义了一个简单的线性布局。接下来我们会在程序中获取该线性布局容器，并往该容器中添加组件。下面是该示例的程序代码。

程序清单：codes\02\2.1\MixView\src\org\crazyit\ui\MixView.java

```
public class MixView extends Activity
{
      // 定义一个访问图片的数组
      int[] images = new int[] {
                  R.drawable.java,
                  R.drawable.ee,
```

```
                R.drawable.classic,
                R.drawable.ajax,
                R.drawable.xml, };
        int currentImg = 0;
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取 LinearLayout 布局容器
            LinearLayout main = (LinearLayout) findViewById(R.id.root);
            // 程序创建 ImageView 组件
            final ImageView image = new ImageView(this);
            // 将 ImageView 组件添加到 LinearLayout 布局容器中
            main.addView(image);
            // 初始化时显示第一张图片
            image.setImageResource(images[0]);
            image.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View v)
                {
                    // 改变 ImageView 里显示的图片
                    image.setImageResource(images[++currentImg % images.length]);
                }
            });
        }
}
```

上面的程序中第一行粗体字代码获取了该 Activity 所显示的 LinearLayout(线性布局管理器)，第 2 和第 3 行粗体字代码用于创建一个 ImageView，并将该 ImageView 添加到 LinearLayout 容器中——其中 LinearLayout 布局管理器通过 XML 布局文件管理，而 ImageView 组件则由 Java 代码管理。

除此之外，上面的程序还为 ImageView 组件添加了一个单击事件。当用户单击该组件时，ImageView 显示下一张图片。运行上面的程序可以看到如图 2.5 所示界面。



图 2.5 XML 文件和 Java 代码混合控制布局

➤➤2.1.5 开发自定义 View

前面已经提到，View 组件的作用类似于 Swing 编程中的 JPanel，它只是一个矩形的空白区域，View 组件没有任何内容。对于 Android 应用的其他 UI 组件来说，它们都继承了 View 组件，然后在 View 组件提供的空白区域上绘制外观。

基于 Android UI 组件的实现原理，开发者完全可以开发出项目定制的组件——当 Android 系统提供的 UI 组件不足以满足项目需要时，开发者可以通过继承 View 来派生自定义组件。

当开发者打算派生自己的 UI 组件时，首先定义一个继承 View 基类的子类，然后重写 View 类的一个或多个方法，通常可以被用户重写的方法如下。

构造器：重写构造器是定制 View 的最基本方式，当 Java 代码创建一个 View 实例，或根据 XML 布局文件加载并构建界面时将需要调用该构造器。

- **onFinishInflate()**：这是一个回调方法，当应用从 XML 布局文件加载该组件并利用它来构建界面之后，该方法将会被回调。
- **onMeasure(int, int)**：调用该方法来检测 View 组件及它所包含的所有子组件的大小。

- onLayout(boolean, int, int, int, int)：当该组件需要分配其子组件的位置、大小时，该方法就会被回调。
- onSizeChanged(int, int, int, int)：当该组件的大小被改变时回调该方法。
- onDraw(Canvas)：当该组件将要绘制它的内容时回调该方法进行绘制。
- onKeyDown(int, KeyEvent)：当某个键被按下时触发该方法。
- onKeyUp(int, KeyEvent)：当松开某个键时触发该方法。
- onTrackballEvent(MotionEvent)：当发生轨迹球事件时触发该方法。
- onTouchEvent(MotionEvent)：当发生触摸屏事件时触发该方法。
- onWindowFocusChanged(boolean)：当该组件得到、失去焦点时触发该方法。
- onAttachedToWindow()：当把该组件放入某个窗口时触发该方法。
- onDetachedFromWindow()：当把该组件从某个窗口上分离时触发该方法。
- onWindowVisibilityChanged(int)：当包含该组件的窗口的可见性发生改变时触发该方法。

当需要开发自定义 View 时，开发者并不需要重写上面列出的所有方法，而是可以根据业务需要重写上面的部分方法，例如下面的示例程序就只重写 onDraw(Canvas)方法。

实例：跟随手指的小球

为了实现一个跟随手指的小球，我们考虑开发自定义的 UI 组件，这个 UI 组件将会在指定位置绘制一个小球，这个位置可以动态改变。当用户通过手指在屏幕上拖动时，程序监听到这个手机动作，并把手指动作的位置传入自定义 UI 组件，并通知该组件重绘即可。

下面是自定义组件的代码。

程序清单：codes\02\2.1\CustomView\src\org\crazyit\ui\DrawView.java

```
public class DrawView extends View
{
    public float currentX = 40;
    public float currentY = 50;
    // 定义、并创建画笔
    Paint p = new Paint();
    public DrawView(Context context)
    {
        super(context);
    }
    public DrawView(Context context , AttributeSet set)
    {
        super(context,set);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 设置画笔的颜色
        p.setColor(Color.RED);
        // 绘制一个小圆（作为小球）
        canvas.drawCircle(currentX, currentY, 15, p);
    }
    // 为该组件的触碰事件重写事件处理方法
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
```

```
            // 修改 currentX、currentY 两个属性
            currentX = event.getX();
            currentY = event.getY();
            // 通知当前组件重绘自己
            invalidate();
            // 返回 true 表明该处理方法已经处理该事件
            return true;
        }
    }
```

上面的 DrawView 组件继承了 View 基类，并重写了 onDraw 方法——该方法负责在该组件的指定位置绘制一个小球。除此之外，该组件还重写了 onTouchEvent(MotionEvent event) 方法，该方法用于处理该组件的触碰事件，当用户手指触碰该组件时将会激发该方法。当手指在触摸屏上移动时，将会不断地触发触摸屏事件，事件监听器中负责触发事件的坐标将被传入 DrawView 组件，并通知该组件重绘——这样即可保证 DrawView 上小球跟随手指移动而移动。

有了这个自定义组件之后，接下来可以通过 Java 代码把该组件添加到指定容器中，这样就可以看到该组件的运行效果。下面是该应用的 Activity 类。

程序清单：codes\02\2.1\CustomView\src\org\crazyit\ui\CustomView.java

```
public class CustomView extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取布局文件中的 LinearLayout 容器
        LinearLayout root = (LinearLayout) findViewById(R.id.root);
        // 创建 DrawView 组件
        final DrawView draw = new DrawView(this);
        // 设置自定义组件的最大宽度、高度
        draw.setMinimumWidth(300);
        draw.setMinimumHeight(500);
        root.addView(draw);
    }
}
```

上面的程序中先创建了自定义组件（DrawView）的实例，然后程序将该组件添加到 LinearLayout 容器中。

运行上面的程序看到结果如图 2.6 所示。



图 2.6 跟随手指的小球

该实例依然在 Java 代码中创建了 DrawView 组件的实例，并将它添加到 LinearLayout 容器中，实际上完全可以在 XML 布局文件中管理该组件，如果我们使用如下布局文件：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/root">
<org.crazyit.ui.DrawView
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

```
</LinearLayout>
```

上面的布局文件已经添加了自定义组件，因此 Java 代码中只要加载该界面布局文件即可，无须通过 Java 代码来添加该自定义组件，因此 Activity 的代码可以简化为如下形式：

```
public class CustomView extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

2.2 第 1 组 UI 组件：布局管理器

Android 的界面组件比较多，如果不理顺它们内在的关系，孤立地学习、记忆这些 UI 组件，不仅学习起来事倍功半，而且不利于掌握它们内在的关系。为了帮助读者更好地掌握 Android 界面组件的关系，本书将会把这些界面组件按照它们的关联分析，分为几组进行介绍。本节介绍的是第一组 UI 组件：以 ViewGroup 为基类派生的布局管理器。

为了更好地管理 Android 应用的用户界面里的各组件，Android 提供了布局管理器。通过使用布局管理器，Android 应用的图形用户界面具有良好的平台无关性。通常来说，推荐使用布局管理器来管理组件的分布、大小，而不是直接设置组件位置和大小。例如通过如下代码定义了一个文本框（TextView）：

```
TextView hello = new TextView(this);
hello.setText("Hello Android");
```

为了让这个组件在不同的手机屏幕上都能运行良好——不同手机屏幕的分辨率、尺寸并不完全相同，如果让程序手动控制每个组件的大小、位置，则将给编程带来巨大的困难。为了解决这个问题，Android 提供了布局管理器。布局管理器可以根据运行平台来调整组件的大小，程序员要做的，只是为容器选择合适的布局管理器。

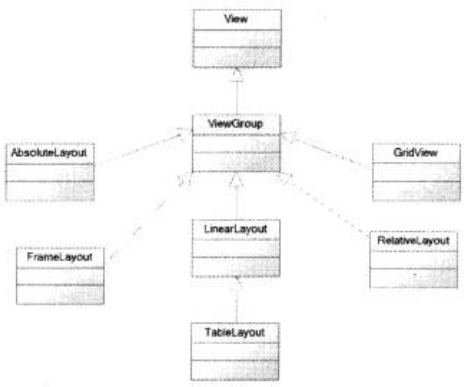


图 2.7　Android 布局管理器的类图

与 Swing 界面编程不同的是，Android 的布局管理器本身就是一个 UI 组件，所有的布局管理器都是 ViewGroup 的子类。图 2.7 显示了 Android 布局管理器的类图。

从图 2.7 可以看出，所有布局都可作为容器类使用，因此可以调用多个重载的 addView()向布局管理器中添加组件。实际上，我们完全可以用一个布局管理器嵌套到其他布局管理器中——因为布局管理器也继承了 View，也可以作为普通 UI 组件使用。

➤➤2.2.1　线性布局

线性布局由 LinearLayout 类来代表，线性布局有点像 Swing 编程里的 Box，它们都会将容器里的组件一个挨着一个地排列起来。LinearLayout 可以控制各组件横向排列（通过设置

android:orientation 属性控制），也可控制各组件纵向排列。

Android 的线性布局不会换行，当组件一个挨着一个地排列到头之后，剩下的组件将不会被显示出来。

表 2.4 显示了 LinearLayout 支持的常用 XML 属性及相关方法的说明。

表 2.4 LineLayout 的常用 XML 属性及相关方法

XML 属性	相关方法	说 明
android:baselineAligned	setBaselineAligned(boolean)	该属性设为 false，将会阻止该布局管理器与它的子元素的基线对齐
android:divider	setDividerDrawable(Drawable)	设置垂直布局时两个按钮之间的分隔条
android:gravity	setGravity(int)	设置布局管理器内组件的对齐方式。该属性支持 top、buttom、left、right、center_vertical、fill_vertical、center_horizontal、fill_horizontal、center、fill、clip_vertical、clip_horizontal 几个属性值。也可以同时指定多种对齐方式的组合，例如 left\|center_vertical 代表出现在屏幕左边，而且垂直居中
android:measureWithLargestChild	setMeasureWithLargestChildEnabled(boolean)	当该属性设为 true 时，所有带权重的子元素都会具有最大子元素的最小尺寸
android:orientation	setOrientation(int)	设置布局管理器内组件的排列方式，可以设置为 horizontal（水平排列）、vertical（垂直排列、默认值）两个值之一

LinearLayout 包含的所有子元素都受 LinearLayout.LayoutParams 控制，因此 LinearLayout 包含的子元素可以额外指定如表 2.5 所示的属性。

表 2.5 LineLayout 子元素支持的常用 XML 属性及相关方法

XML 属性	说 明
android:layout_gravity	指定该子元素在 LinearLayout 中的对齐方式
android:layout_weight	指定该子元素在 LinearLayout 中所占的权重



提示：

基本上很多布局管理器都提供了相应的 LayoutParams 内部类，该内部类用于控制它们的子元素支持指定 android:layout_gravity 属性，该属性设置该子元素在父容器中的对齐方式。与 android:layout_gravity 相似的属性还有 android:gravity 属性（一般容器才支持指定该属性），android:gravity 属性用于控制它所包含的子元素的对齐方式。

例如定义如下 XML 布局管理器：

程序清单：codes\02\2.2\LinearLayoutTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="bottom|center_horizontal"
        >
    <Button
        android:id="@+id/bn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn1"
        />
    <Button
        android:id="@+id/bn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn2"
        />
    <Button
        android:id="@+id/bn3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn3"
        />
    <Button
        android:id="@+id/bn4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn4"
        />
    <Button
        android:id="@+id/bn5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bn5"
        />
    </LinearLayout>
```

上面的界面布局非常简单，它只是定义了一个简单的线性布局，并在线性布局中定义了 5 个按钮。定义线性布局时使用粗体字代码指定了垂直排列所有组件，而且所有组件对齐到容器底部并且水平居中，运行上面的程序，使用 Activity 显示上面的界面布局，将看到如图 2.8 所示界面。

如果将上面的布局文件中 android:gravity="bottom|center_ horizontal"改为 android:gravity="right|center_vertical"——也就是所有组件水平右对齐、垂直居中，再次使用 Activity 显示该界面布局将看到如图 2.9 所示界面。



图 2.8　垂直的线性布局，底部、居中对齐



图 2.9　垂直的线性布局，水平居右、垂直居中对齐

如果我们将上面的线性布局的方向改为水平，也就是设置 android:orientation="horizontal"，并设置 gravity="top"，再次使用 Activity 来显示该界面布局将看到如图 2.10 所示界面。

从图 2.10 所示的运行结果可以看出，当采用线性布局来管理 5 个按钮组件时，由于这 5 个按钮组件无法在一行中同时显示出来，但 LinearLayout 不会换行显示多余的组件，因此图 2.10 中看不到第 4 和第 5 个按钮。



图 2.10 水平的线性布局，顶端对齐

2.2.2 表格布局

表格布局由 TableLayout 所代表，TableLayout 继承了 LinearLayout，因此它的本质依然是线性布局管理器。表格布局采用行、列的形式来管理 UI 组件，TableLayout 并不需要明确地声明包含多少行、多少列，而是通过添加 TableRow、其他组件来控制表格的行数和列数。

每次向 TableLayout 中添加一个 TableRow，该 TableRow 就是一个表格行，TableRow 也是容器，因此它也可以不断地添加其他组件，每添加一个子组件该表格就增加一列。

如果直接向 TableLayout 中添加组件，那么这个组件将直接占用一行。

在表格布局中，列的宽度由该列中最宽的那个单元格决定，整个表格布局的宽度则取决于父容器的宽度（默认总是占满父容器本身）。

在表格布局管理器中，可以为单元格设置如下三种行为方式。

- **Shrinkable**：如果某个列被设为 Shrinkable，那么该列的所有单元格的宽度可以被收缩，以保证该表格能适应父容器的宽度。
- **Stretchable**：如果某个列被设为 Stretchable，那么该列的所有单元格的宽度可以被拉伸，以保证组件能完全填满表格空余空间。
- **Collapsed**：如果某个列被设为 Collapsed，那么该列的所有单元格会被隐藏。

TableLayout 继承了 LinearLayout，因此它完全可以支持 LinearLayout 所支持的全部 XML 属性，除此之外，TableLayout 还支持如表 2.6 所示的 XML 属性。

表 2.6 TableLayout 的常用 XML 属性及相关方法

XML 属性	相关方法	说　明
android:collapseColumns	setColumnCollapsed(int,boolean)	设置需要被隐藏的列的列序号。多个列序号之间用逗号隔开
android:shrinkColumns	setShrinkAllColumns(boolean)	设置允许被收缩的列的列序号。多个列序号之间用逗号隔开
android:stretchColumns	setStretchAllColumns(boolean)	设置允许被拉伸的列的列序号。多个列序号之间用逗号隔开

实例：丰富的表格布局

下面的程序示范了如何使用 TableLayout 来管理组件的布局，下面是界面布局所使用的布局文件。

程序清单：codes\02\2.2\TableLayoutTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
```

```
<!-- 定义第一个表格布局，指定第 2 列允许收缩，第 3 列允许拉伸 -->
<TableLayout android:id="@+id/TableLayout01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:shrinkColumns="1"
      android:stretchColumns="2"
>
...<!-- 表格内容接下来详细讲解 -->
</TableLayout>
<!-- 定义第 2 个表格布局 ，指定第 2 列隐藏-->
<TableLayout android:id="@+id/TableLayout01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:collapseColumns="1"
>
...<!-- 表格内容接下来详细讲解 -->
</TableLayout>
<!-- 定义第 3 个表格布局，指定第 2 列和第 3 列可以被拉伸-->
<TableLayout android:id="@+id/TableLayout01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:stretchColumns="1,2"
>
...<!-- 表格内容接下来详细讲解 -->
</TableLayout>
</LinearLayout>
```

上面页面中定义了三个 TableLayout，三个 TableLayout 中粗体字代码指定了它们对各列的控制行为：

- 第一个 TableLayout，指定第 2 列允许收缩，第 3 列允许拉伸。
- 第二个 TableLayout，指定第 2 列被隐藏。
- 第三个 TableLayout，指定第 2 列和第 3 列允许拉伸。

接下来为布局中第一个 TableLayout 添加两行，第一行不使用 TableRow，直接添加一个 Button，那么该 Button 自己将占用整行。第二行先添加一个 TableRow，并为 TableRow 添加三个 Button，那说明该表格将包含三列。第一个表格的界面布局代码如下。

程序清单：codes\02\2.2\TableLayoutTest\res\layout\main.xml

```
<!-- 直接添加按钮，它自己会占一行 -->
<Button android:id="@+id/ok1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="独自一行的按钮"
      />
<!-- 添加一个表格行 -->
<TableRow>
<!-- 为该表格行添加三个按钮 -->
<Button android:id="@+id/ok2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="普通按钮"
      />
<Button android:id="@+id/ok3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="收缩的按钮"
      />
<Button android:id="@+id/ok4"
      android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="拉伸的按钮"
        />
</TableRow>
```

注意：

在上面的界面布局文件中我们直接把按钮上的文本写在布局文件中，这不是一种好的做法，因为 Android 推荐将这些字符串集中放到 XML 文件中管理。但此处为了编程简单，所以直接在 XML 布局文件中给出了按钮文本的字符串。



接下来为布局中第 2 个 TableLayout 添加两行，第 1 行不使用 TableRow，直接添加一个 Button，那么该 Button 自己将占用整行。第 2 行先添加一个 TableRow，并为 TableRow 添加三个 Button，那说明该表格将包含三列。第 2 个表格内容与第 1 个表格内容基本相似，但由于我们为第 2 个表格指定了 android:collapseColumns="1"，这意味着第 2 行中间的按钮将会被隐藏。

接下来为布局中第 3 个 TableLayout 添加三行，第 1 行不使用 TableRow，直接添加一个 Button，那么该 Button 自己将占用整行。第 2 行先添加一个 TableRow，并为 TableRow 添加三个 Button，那说明该表格将包含三列。第 3 行也先添加一个 TableRow，并为 TableRow 添加两个按钮，这意味着表格的第 3 行只有两个单元格，第 3 个单元格为空。

第 3 个表格布局管理器的内容如下。

程序清单：codes\02\2.2\TableLayoutTest\res\layout\main.xml

```
<!-- 直接添加按钮，它自己会占一行 -->
<Button android:id="@+id/ok9"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="独自一行的按钮"
        />
<!--定义一个表格行-->
<TableRow>
<!-- 为该表格行添加三个按钮 -->
<Button android:id="@+id/ok10"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="普通按钮"
        />
<Button android:id="@+id/ok11"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拉伸的按钮"
        />
<Button android:id="@+id/ok12"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=拉伸的按钮"
        />
</TableRow>
<!--定义一个表格行-->
<TableRow>
<!-- 为该表格行添加两个按钮 -->
<Button android:id="@+id/ok13"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="普通按钮"
        />
    <Button android:id="@+id/ok14"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="拉伸的按钮"
        />
</TableRow>
```

使用 Activity 来显示上面的界面布局将会看到如图 2.11 所示界面。



图 2.11　表格布局

2.2.3　帧布局

帧布局由 FrameLayout 所代表，FrameLayout 直接继承了 ViewGroup 组件。

帧布局容器为每个加入其中的组件创建一个空白的区域（称为一帧），每个子组件占据一帧，这些帧都会根据 gravity 属性执行自动对齐。帧布局的效果有点类似于 AWT 编程的 CardLayout，都是把组件一个一个地叠加在一起。与 CardLayout 的区别在于，CardLayout 可以将下面的 Card 移上来，但 FrameLayout 则没有提供相应的方法。

表 2.7 显示了 FrameLayout 常用的 XML 属性及相关方法说明。

表 2.7　FrameLayout 的常用 XML 属性及相关方法

XML 属性	相关方法	说　明
android:foreground	setForeground(Drawable)	设置该帧布局容器的前景图像
android:foregroundGravity	setForegroundGravity(int)	定义绘制前景图像的 gravity 属性

FrameLayout 包含的子元素也受 FrameLayout.LayoutParams 控制，因此它所包含的子元素也可指定 android:layout_gravity 属性，该属性控制该子元素在 FrameLayout 中的对齐方式。

下面示范了帧布局的用法，可以看到 6 个 TextView 叠加在一起，上面的 TextView 遮住下面的 TextView。下面是使用帧布局的页面定义代码。

程序清单：codes\02\2.2\FrameLayoutTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 依次定义 6 个 TextView，先定义的 TextView 位于底层
    后定义的 TextView 位于上层 -->
<TextView android:id="@+id/view01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:width="320px"
    android:height="320px"
    android:background="#f00"
    />
<TextView android:id="@+id/view02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:width="280px"
```

```
        android:height="280px"
        android:background="#0f0"
        />
    <TextView android:id="@+id/view03"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="240px"
        android:height="240px"
        android:background="#00f"
        />
    <TextView android:id="@+id/view04"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="200px"
        android:height="200px"
        android:background="#ff0"
        />
    <TextView android:id="@+id/view05"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="160px"
        android:height="160px"
        android:background="#f0f"
        />
    <TextView android:id="@+id/view06"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:width="120px"
        android:height="120px"
        android:background="#0ff"
        />
    </FrameLayout>
```

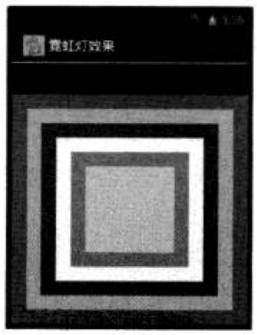


图 2.12 帧布局

上面的界面布局定义使用 FrameLayout 布局，并向该布局容器中添加了 7 个 TextView，这 7 个 TextView 的高度完全相同，而宽度则逐渐减少——这样可以保证最先添加的 TextView 不会被完全遮挡；而且我们设置了 7 个 TextView 的背景色渐变。

使用 Activity 显示上面的界面布局，将看到如图 2.12 所示效果。

实例：霓虹灯效果

如果考虑轮换改变上面的帧布局中 6 个 TextView 的背景色，就会看到上面的颜色渐变条不断地变换，就像大街上的霓虹灯一样。下面的程序还是使用上面的 FrameLayout 布局管理器，只是程序启动了一条线程来控制周期性地改变这 6 个 TextView 的背景色。下面是该主程序的代码。

程序清单：codes\02\2.2\FrameLayoutTest\src\org\crazyit\ui\FrameLayoutTest.java

```
public class FrameLayoutTest extends Activity
{
    private int currentColor = 0;
    // 定义一个颜色数组
    final int[] colors = new int[] {
        R.color.color1,
```

```
            R.color.color2,
            R.color.color3,
            R.color.color4,
            R.color.color5,
            R.color.color6
        };
        final int[] names = new int[] {
            R.id.view01,
            R.id.view02,
            R.id.view03,
            R.id.view04,
            R.id.view05,
            R.id.view06 };
        TextView[] views = new TextView[names.length];
        Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 表明消息来自本程序所发送
                if (msg.what == 0x123)
                {
                    for (int i = 0; i < names.length; i++)
                    {
                        views[i].setBackgroundResource(colors[(i
                                + currentColor)      % names.length]);
                    }
                    currentColor++;
                }
                super.handleMessage(msg);
            }
        };
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            for (int i = 0; i < names.length; i++)
            {
                views[i] = (TextView) findViewById(names[i]);
            }
            // 定义一个线程周期性地改变 currentColor 变量值
            new Timer().schedule(new TimerTask()
            {
                @Override
                public void run()
                {
                    // 发送一条空消息通知系统改变 6 个 TextView 组件的背景色
                    handler.sendEmptyMessage(0x123);
                }
            }, 0, 200);
        }
}
```

上面的程序中粗体字代码定义了一个每 0.2 秒执行一次的任务，该任务仅仅向 Handler 发送一条消息，通知它更新 6 个 TextView 的背景色。

可能会有读者提出疑问：为何不直接在 run()方法里直接更新 6 个 TextView 的背景色呢？这是因为 Android 的 View 和 UI 组件不是线程安全的，所以 Android 不允许开发者启动线程访问用户界面的 UI 组件。所以程序中额外定义了一个 Handler 来处理 TextView 背景色的更新。



上面的程序中直接使用了 R.color.color1、R.color.color2、R.color.color3 等整型常量来代表颜色，这也得益于 Android 的资源访问支持，本书后面会有关于颜色资源的详细介绍。



简单地说，上面的程序通过任务调度控制了每间隔 0.2 秒轮换更新一次 6 个 TextView 的背景色，这样看上去就像大街上的“霓虹灯”了。

2.2.4 相对布局

相对布局由 RelativeLayout 代表，相对布局容器内子组件的位置总是相对兄弟组件、父容器来决定的，因此这种布局方式被称为相对布局。

如果 A 组件的位置是由 B 组件的位置来决定的，Android 要求先定义 B 组件，再定义 A 组件。

RelativeLayout 可支持如表 2.8 所示的两个 XML 属性。

表 2.8 RelativeLayout 的 XML 属性及相关方法说明

XML 属性	相关方法	说 明
android:gravity	setGravity(int)	设置该布局容器内各子组件的对齐方式
android:ignoreGravity	setIgnoreGravity(int)	设置哪个组件不受 gravity 属性的影响

为了控制该布局容器中各子组件的布局分布，RelativeLayout 提供了一个内部类：RelativeLayout.LayoutParams，该类提供了大量的 XML 属性来控制 RelativeLayout 布局容器中子组件的布局分布。

RelativeLayout.LayoutParams 里只能设为 true、false 的 XML 属性如表 2.9 所示。

表 2.9 RelativeLayout.LayoutParams 里只能设为 boolean 值的属性

android:layout_centerHorizontal	控制该子组件是否位于布局容器的水平居中
android:layout_centerVertical	控制该子组件是否位于布局容器的垂直居中
android:layout_centerInParent	控制该子组件是否位于布局容器的中央位置
android:layout_alignParentBottom	控制该子组件是否与布局容器底端对齐
android:layout_alignParentLeft	控制该子组件是否与布局容器左边对齐
android:layout_alignParentRight	控制该子组件是否与布局容器右边对齐
android:layout_alignParentTop	控制该子组件是否与布局容器顶端对齐

RelativeLayout.LayoutParams 里属性值为其他 UI 组件 ID 的 XML 属性如表 2.10 所示。

表 2.10 RelativeLayout.LayoutParams 里只能设为其他 UI 组件 ID 的属性

android:layout_toRightOf	控制该子组件位于给出 ID 组件的右侧
android:layout_toLeftOf	控制该子组件位于给出 ID 组件的左侧
android:layout_above	控制该子组件位于给出 ID 组件的上方
android:layout_below	控制该子组件位于给出 ID 组件的下方
android:layout_alignTop	控制该子组件位于给出 ID 组件的上边界对齐

续表

android:layout_alignBottom	控制该子组件位于给出 ID 组件的下边界对齐
android:layout_alignLeft	控制该子组件位于给出 ID 组件的左边界对齐
android:layout_alignRight	控制该子组件位于给出 ID 组件的右边界对齐

除此之外，RelativeLayout.LayoutParams 还继承了 android.view.ViewGroup.MarginLayoutParams，因此 RelativeLayout 布局容器中每个子组件也可指定 android.view.ViewGroup.MarginLayoutParams 所支持的各 XML 属性。

下面以一个示例来介绍相对布局的用法。

实例：梅花布局效果

相对布局容器中的子组件总是相对其他组件来决定分布位置的，可以考虑先把一个组件放在相对布局容器的中间，然后以该组件为中心，将其他组件分布在该组件的四周，这样就可以形成“梅花布局”效果。

下面是“梅花”布局效果的界面布局文件。

程序清单：codes\02\2.2\RelativeLayoutTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 定义该组件位于父容器中间 -->
<TextView
      android:id="@+id/view01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/leaf"
      android:layout_centerInParent="true"
      />
<!-- 定义该组件位于 view01 组件的上方 -->
<TextView
      android:id="@+id/view02"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/leaf"
      android:layout_above="@id/view01"
      android:layout_alignLeft="@id/view01"
      />
<!-- 定义该组件位于 view01 组件的下方 -->
<TextView
      android:id="@+id/view03"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/leaf"
      android:layout_below="@id/view01"
      android:layout_alignLeft="@id/view01"
      />
<!-- 定义该组件位于 view01 组件的左边 -->
<TextView
      android:id="@+id/view04"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
        android:background="@drawable/leaf"
        android:layout_toLeftOf="@id/view01"
        android:layout_alignTop="@id/view01"
        />
    <!-- 定义该组件位于 view01 组件的右边 -->
    <TextView
        android:id="@+id/view05"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/leaf"
        android:layout_toRightOf="@id/view01"
        android:layout_alignTop="@id/view01"
        />
</RelativeLayout>
```

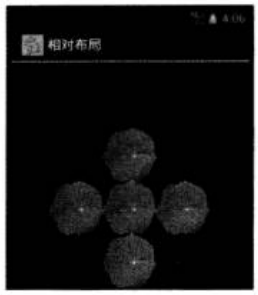


图 2.13 相对布局实现的“梅花效果”

上面的程序中第一行粗体字代码控制该组件位于父容器的中央，接下来定义的 4 个组件依次环绕在第一个组件四周。使用 Activity 来显示上面的布局文件可以看到如图 2.13 所示效果。



提示：

这里例子中使用了一张图片，这也是利用了 Android 的图片资源访问策略，本书后面还会集中介绍图片资源访问。

2.2.5 Android 4.0 新增的网格布局

网格布局由 GridLayout 代表，它是 Android 4.0 新增的布局管理器，因此需要在 Android 4.0 之后的版本中才能使用该布局管理器。如果希望在更早的 Android 平台上使用该布局管理器，则需要导入相应的支撑库。

GridLayout 的作用类似于 HTML 中的 table 标签，它把整个容器划分成 rowsxcolumns 个网格，每个网格可以放置一个组件。除此之外，也可以设置一个组件横跨多少列、一个组件纵跨多少行。

GridLayout 提供了 setRowCount(int)和 setColumnCount(int)方法来控制该网格的行数量和列数量。

表 2.11 显示了 GridLayout 常用的 XML 属性及相关方法。

表 2.11 RelativeLayout 的 XML 属性及相关方法说明

XML 属性	相关方法	说 明
android:alignmentMode	setAlignmentMode(int)	设置该布局管理器采用的对齐模式
android:columnCount	setColumnCount(int)	设置该网格的列数量
android:columnOrderPreserved	setColumnOrderPreserved(boolean)	设置该网格容器是否保留列序号
android:rowCount	setRowCount(int)	设置该网格的行数量
android:rowOrderPreserved	setRowOrderPreserved(boolean)	设置该网格容器是否保留行序号
android:useDefaultMargins	setUseDefaultMargins(boolean)	设置该布局管理器是否使用默认的页边距

为了控制 GridLayout 布局容器中各子组件的布局分布，GridLayout 提供了一个内部类：GridLayout.LayoutParams，该类提供了大量的 XML 属性来控制 GridLayout 布局容器中子组件的布局分布。

表 2.12 显示了 GridLayout.LayoutParams 常用的 XML 属性及相关方法。

表 2.12　GridLayout.LayoutParams 的 XML 属性及相关方法说明

XML 属性	相关方法	说　明
android:layout_column		设置该子组件在 GridLayout 的第几列
android:layout_columnSpan		设置该子组件在 GridLayout 横向上跨几列
android:layout_gravity	setGravity(int)	设置该子组件采用何种方式占据该网格的空间
android:layout_row		设置该子组件在 GridLayout 的第几行
android:layout_rowSpan		设置该子组件在 GridLayout 纵向上跨几行

下面将会通过一个实例来示范 GridLayout 的功能和用法。

实例：计算器界面



图 2.14　计算器界面

为了实现如图 2.14 所示的计算器界面，可以考虑将该界面分解成一个 6×4 的网格，其中第一个文本框横跨 4 列，第二个按钮横跨 4 列，后面每个按钮各占一格。

为了实现该界面，考虑按如下步骤来实现该界面：

① 在布局管理器中定义一个 GridLayout，并在该 GridLayout 中依次定义文本框、按钮，该文本框、按钮各横跨 4 列。

② 在 Java 代码中循环 16 次，依次添加 16 个按钮。

下面先定义如下界面文件：

```
<?xml version="1.0" encoding="utf-8" ?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:rowCount="6"
      android:columnCount="4"
      android:id="@+id/root"
      >
<!-- 定义一个横跨 4 列的文本框，
并设置该文本框的前景色、背景色等属性  -->
<TextView
    android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_columnSpan="4"
      android:textSize="50sp"
      android:layout_marginLeft="4px"
      android:layout_marginRight="4px"
      android:padding="5px"
      android:layout_gravity="right"
      android:background="#eee"
      android:textColor="#000"
      android:text="0"/>
<!-- 定义一个横跨 4 列的按钮 -->
<Button
    android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_columnSpan="4"
      android:text="清除"/>
</GridLayout>
```

上面的界面布局文件中定义一个 6×4 的 GridLayout，并在该布局管理器中添加了两个子组件，其中第一个子组件横跨 4 列，第二个子组件也横跨 4 列。

接下来使用如下 Java 代码采用循环控制添加 16 个按钮。

程序清单：codes\02\2.2\GridLayoutTest\src\org\crazyit\ui\GridLayoutTest.java

```
public class GridLayoutTest extends Activity
{
    GridLayout gridLayout;
    // 定义16个按钮的文本
    String[] chars = new String[]
        {
            "7" , "8" , "9" , "÷",
            "4" , "5" , "6" , "×",
            "1" , "2" , "3" , "-",
            "." , "0" , "=" , "+"
        };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        gridLayout = (GridLayout) findViewById(R.id.root);
        for(int i = 0 ; i < chars.length ; i++)
        {
            Button bn = new Button(this);
            bn.setText(chars[i]);
            // 设置该按钮的字号大小
            bn.setTextSize(40);
            // 指定该组件所在的行
            GridLayout.Spec rowSpec = GridLayout.spec(i / 4 + 2);
            // 指定该组件所在的列
            GridLayout.Spec columnSpec = GridLayout.spec(i % 4);
            GridLayout.LayoutParams params = new GridLayout.LayoutParams(
                    rowSpec , columnSpec);
            // 指定该组件占满父容器
            params.setGravity(Gravity.FILL);
            gridLayout.addView(bn , params);
        }
    }
}
```

上面的粗体字代码采用循环向 GridLayout 中添加了 16 个按钮，添加 16 个按钮时指定了每个按钮所在的行号、列号，并指定这些按钮将会自动填充单元格的所有空间——这样避免单元格中的大量空白。运行该程序将可以看到如图 2.14 所示界面。

2.2.6 绝对布局

绝对布局由 AbsoluteLayout 代表。绝对布局就像 Java AWT 编程中的空布局，就是 Android 不提供任何布局控制，而是由开发人员自己通过 *X* 坐标、*Y* 坐标来控制组件的位置。当使用 AbsoluteLayout 作为布局容器时，布局容器不再管理子组件的位置、大小——这些都需要开发人员自己控制。

注意：

大部分时候，使用绝对布局都不是一个好思路，因为运行 Android 应用的手机往往千差万别，因此屏幕大小、分辨率都可能存在较大差异，使用绝对布局会很难兼顾不同屏幕大小、分辨率的问题。因此 AbsoluteLayout 布局管理器已经过时。

使用绝对布局时，每个子组件都可指定如下两个 XML 属性。

- layout_x：指定该子组件的 X 坐标。
- layout_y：指定该子组件的 Y 坐标。

实例：登录界面

下面介绍一个使用绝对布局开发的登录界面的实例，这个登录界面中所有组件都通过“绝对定位”的方式来指定位置。下面是该登录界面的界面布局文件。

程序清单：codes\02\2.2\AbsoluteLayoutTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 定义一个文本框，使用绝对定位 -->
<TextView
      android:layout_x="20dip"
      android:layout_y="20dip"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="用户名："
      />
<!-- 定义一个文本编辑框，使用绝对定位 -->
<EditText
      android:layout_x="80dip"
      android:layout_y="15dip"
      android:layout_width="wrap_content"
      android:width="200px"
      android:layout_height="wrap_content"
      />
<!-- 定义一个文本框，使用绝对定位 -->
<TextView
      android:layout_x="20dip"
      android:layout_y="80dip"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="密　码："
      />
<!-- 定义一个文本编辑框，使用绝对定位 -->
<EditText
      android:layout_x="80dip"
      android:layout_y="75dip"
      android:layout_width="wrap_content"
      android:width="200px"
      android:layout_height="wrap_content"
      android:password="true"
      />
<!-- 定义一个按钮，使用绝对定位 -->
<Button
      android:layout_x="130dip"
      android:layout_y="135dip"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="登　录"
      />
</AbsoluteLayout>
```

上面的绝对布局容器中的每个子组件都指定了 layout_x、layout_y 两个定位属性，这样才控制了每个子组件在容器中的出现位置。使用 Activity 显示上面的页面，将看到如图 2.15 所示界面。



图 2.15 绝对布局实现的登录界面

不要以为笔者一下就通过绝对布局做出了图 2.15 所示的登录界面，实际上这个登录界面是不断调整各组件的位置，经过多次尝试之后得到的结果。当使用绝对布局来控制子组件布局时，编程要烦琐得多，而且在不同屏幕上的显示效果差异也很大。

上面的界面布局中指定各组件的 android:layout_x、android:layout_y 属性时指定了形如 20dip 这样的属性值，这是一个距离值。Android 中一般支持如下常用的距离单位。

- px（像素）：每个 px 对应屏幕上的一个点。
- dip 或 dp（device independent pixels，设备独立像素）：一种基于屏幕密度的抽象单位。在每英寸 160 点的显示器上，1dip = 1px。但随着屏幕密度的改变，dip 与 px 的换算会发生改变。
- sp（scaled pixels，比例像素）：主要处理字体的大小，可以根据用户的字体大小首选项进行缩放。
- in（英寸）：标准长度单位。
- mm（毫米）：标准长度单位。
- pt（磅）：标准长度单位，1/72 英寸。

2.3 第 2 组 UI 组件：TextView 及其子类

前面介绍了 Android 界面编程的一些基础知识，接下来将要介绍的是 Android 基本界面组件。“九层之台，起于垒土” ——无论看上去多么美观的 UI 界面，开始都是先创建容器（ViewGroup 的实例），然后不断地向容器中添加界面组件，最后形成一个美观的 UI 界面。掌握这些基本用户界面组件是学好 Android 编程的基础。

2.3.1 文本框（TextView）与编辑框（EditText）的功能和用法

TextView 直接继承了 View，它还是 EditText、Button 两个 UI 组件类的父类。TextView 的作用就是在界面上显示文本——从这个意义上来看，它有点类似于 Swing 编程中的 JLabel，不过它比 JLabel 功能更强大。

从功能上来看，TextView 其实就是一个文本编辑器，只是 Android 关闭了它的文字编辑功能。如果开发者想要定义一个可以编辑内容的文本框，则可以使用它的子类：EditText，EditText 允许用户编辑文本框中的内容。

TextView 还派生了一个 CheckedTextView，CheckedTextView 增加了一个 checked 状态，开发者可通过 setChecked(boolean)和 isChecked()方法来改变、访问该组件的 checked 状态。除此之外，该组件还可通过 setCheckMarkDrawable()方法来设置它的勾选图标。

不仅如此，TextView 还派生出了 Button 类，TextView 类及其子类的类图如图 2.16 所示。

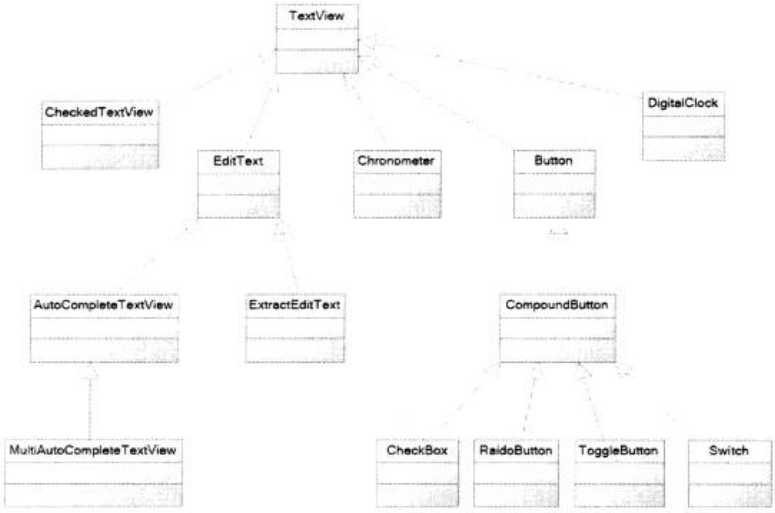


图 2.16　TextView 及其子类的类图

TextView 和 EditText 具有很多相似之处，它们之间的最大区别在于 TextView 不允许用户编辑文本内容，而 EditText 则允许用户编辑文本内容。

TextView 提供了大量的 XML 属性，这些 XML 属性大部分不仅可适用于 TextView，而且可适用于它的子类（EditText、Button 等）。表 2.13 显示了 TextView 支持的 XML 属性及相关方法的说明。

表 2.13　TextView 的 XML 属性及相关方法的说明

XML 属性	相关方法	说　明
android:autoLink	setAutoLinkMask(int)	是否将符合指定格式的文本转换为可单击的超链接形式
android:autoText	setKeyListener(KeyListener)	控制是否将 URL、E-mail 地址等链接自动转换为可单击的链接
android:capitalize	setKeyListener(KeyListener)	控制是否将用户输入的文本转换为大写字母。该属性支持如下属性值。 ➢ none：不转换 ➢ sentences：每个句子的首字母大写 ➢ words：每个单词首字母大写 ➢ characters：每个字母都大写
android:cursorVisible	setCursorVisible(boolean)	设置该文本框的光标是否可见
android:digits	setKeyListener(KeyListener)	如果该属性设为 true，则该文本框对应一个数字输入方法，并且只接受那些合法字符
android:drawableBottom	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的底端绘制指定图像
android:drawableEnd		在文本框内文本的结尾处绘制指定图像
android:drawableLeft	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的左边绘制指定图像
android:drawablePadding	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	设置文本框内文本与图形之间的间距

续表

XML 属性	相关方法	说 明
android:drawableRight	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的右边绘制指定图像
android:drawableStart		在文本框内文本的开始处绘制指定图像
android:drawableTop	setCompoundDrawablesWithIntrinsicBounds (Drawable,Drawable,Drawable,Drawable)	在文本框内文本的顶端绘制指定图像
android:editable		设置该文本是否允许编辑
android:ellipsize	setEllipsize(TextUitls.TruncateAt)	设置当显示的文本超过了 TextView 的长度时如何处理文本内容。该属性支持如下属性值。 ➢ none：不做任何处理 ➢ start：在文本开始处截断，并显示省略号 ➢ middle：在文本中间处截断，并显示省略号 ➢ end：在文本结尾处截断，并显示省略号 ➢ marquee：使用 marquee 滚动动画显示文本
android:ems	setEms(int)	设置该组件的宽度，以 em 为单位
android:fontFamily	setTypeface(Typeface)	设置该文本框内文本的字体
android:gravity	setGravity(int)	设置文本框内文本的对齐方式
android:height	setHeight(int)	设置该文本框的高度（以 pixel 为单位）
android:hint	setHint(int)	设置当该文本框内容为空时，文本框内默认显示的提示文本
android:imeActionId	setImeActionLabel(CharSequence , int)	当该文本框关联输入法时，为输入法提供 EditorInfo.actionId 值
android:imeActionLabel	setImeActionLabel(CharSequence , int)	当该文本框关联输入法时，为输入法提供 EditorInfo.actionLabel 值
android:imeOptions	setImeOptions(int)	当该文本框关联输入法时，为输入法指定额外的选项
android:includeFontPadding	setIncludeFontPadding(boolean)	设置是否为字体保留足够的空间。默认为 true
android:inputMethod	setKeyListener(KeyListener)	为该文本框指定特定的输入法。该属性值为输入法的全限定类名
android:inputType	setRawInputType(int)	指定该文本框的类型。该属性有点类似于 HTML 中<input.../>元素的 type 属性。该属性支持大量属性值，不同属性值用于指定特定的输入框
android:lineSpacingExtra	setLineSpacing(float,float)	控制两行文本之间的额外间距。与 android:lineSpacingMultiplier 属性结合使用
android:lineSpacingMultiplier	setLineSpacing(float,float)	控制两行文本之间的额外间距。每行文本为高度*该属性值+ android:lineSpacingExtra 属性值
android:lines	setLines(int)	设置该文本框默认占几行
android:linksClickable	setLinksClickable(boolean)	控制该文本框的 URL、E-mail 等链接是否可点击
android:marqueeRepeatLimit	setMarqueeRepeatLimit(int)	设置 marquee 动画重复的次数
android:maxEms	setMaxEms(int)	指定该文本框的最大宽度（以 em 为单位）
android:maxHeight	setMaxHeight(int)	指定该文本框的最大高度（以 pixel 为单位）
android:maxLength	setFilters(InputFilter)	设置该文本框的最大字符长度

续表

XML 属性	相关方法	说　明
android:maxLines	setMaxLines(int)	设置该文本框最多占几行
android:maxWidth	setMaxWidth(int)	指定该文本框的最大宽度（以 pixel 为单位）
android:minEms	setMinEms(int)	指定该文本框的最小宽度（以 em 为单位）
android:minHeight	setMinHeight(int)	指定该文本框的最小高度（以 pixel 为单位）
android:minLines	setMinLines(int)	设置该文本框最少占几行
android:minWidth	setMinWidth(int)	指定该文本框的最小宽度（以 pixel 为单位）
android:numeric	setKeyListener(KeyListener)	设置该文本框关联的数值输入法。该属性值支持如下属性值。 ➢ integer：指定关联整数输入法 ➢ signed：允许输入符号的数值输入法 ➢ decimal：允许输入小数点的数值输入法
android:password	setTransformationMethod(TransformationMethod)	设置该文本框是一个密码框（以点代替字符）
android:phoneNumber	setKeyListener(KeyListener)	设置该文本框只能接受电话号码
android:privateImeOptions	setPrivateImeOptions(String)	
android:scrollHorizontally	setHorizontallyScrolling(boolean)	设置当该文本框不够显示全部内容时是否允许水平滚动
android:selectAllOnFocus	setSelectAllOnFocus(boolean)	如果文本框的内容可选择，设置是否当它获得焦点时自动选中所有文本
android:shadowColor	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影的颜色
android:shadowDx	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影在水平方向的偏移
android:shadowDy	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影在垂直方向的偏移
android:shadowRadius	setShadowLayer(float,float,float,int)	设置文本框内文本的阴影的模糊程度。该值越大，阴影越模糊
android:singleLine	setTransformationMethod	设置该文本框是否为单行模式。如果设为 true，文本框不会换行
android:text	setText(CharSequence)	设置文本框内文本的内容
android:textAllCaps	setAllCaps(boolean)	设置是否将文本框的所有字母显示为大写字母
android:textAppearance		设置该文本框的颜色、字体、大小等样式
android:textColor	setTextColor(ColorStateList)	设置文本框中文本的颜色
android:textColorHighlight	setHighlightColor(int)	设置文本框中文本被选中时的颜色
android:textColorHint	setHintTextColor(int)	设置文本框中提示文本的颜色
android:textColorLink	setLinkTextColor(int)	设置该文框中链接的颜色
android:textIsSelectable	isTextSelectedable()	设置该文本框不能编辑时，文本框内的文本是否可以被选中
android:textScaleX	setTextScaleX(float)	设置文本框内文本在水平方向上的缩放因子
android:textSize	setTextSize(float)	设置文本框内文本的字号大小
android:textStyle	setTypeface(Typeface)	设置文本框内文本的字体风格，如粗体、斜体等
android:typeface	setTypeface(Typeface)	设置文本框内文本的字体风格
android:width	setWidth(int)	设置该文本框的宽度（以 pixel 为单位）

下面通过系列实例来介绍 TextView 和 CheckedTextView 的用法。

实例：不同颜色、字体、带链接的文本

由于 TextView 提供了大量 XML 属性，因此我们可以通过这些 XML 属性来控制 TextView 中文本的行为，例如如下界面布局文件。

程序清单：codes\02\2.3\TextViewTest\res\layout \main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <!-- 设置字号为 20pt，文本框结尾处绘制图片 -->
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="我爱 Java"
    android:textSize="20pt"
    android:drawableEnd="@drawable/ic_launcher"
    />
    <!-- 设置中间省略，所有字母大写 -->
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:text="我爱 Java 我爱 Java 我爱 Java 我爱 Java 我爱 Java 我 aaaJava"
    android:ellipsize="middle"
    android:textAllCaps="true"
    />
    <!-- 对邮件、电话增加链接 -->
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:singleLine="true"
    android:text="邮件是 kongyeeku@163.com，电话是 02088888888"
    android:autoLink="email|phone"
    />
    <!-- 设置文字颜色、大小，并使用阴影 -->
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="测试文字"
    android:shadowColor="#00f"
    android:shadowDx="10.0"
    android:shadowDy="8.0"
    android:shadowRadius="3.0"
    android:textColor="#f00"
    android:textSize="18pt"
    />
    <!-- 测试密码框 -->
    <TextView android:id="@+id/passwd"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:password="true"
    />
    <CheckedTextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="可勾选的文本"
        android:checkMark="@drawable/ok" />
</LinearLayout>
```

上面的界面布局文件中定义了 6 个 TextView，它们指定的规则如下。

- 第 1 个 TextView 指定 android:textSize="20pt"，这就指定了字号为 20pt。且指定了在文本框的结尾处绘制图片。
- 第 2 个 TextView 指定 android:ellipsize="middle"，这就指定了当文本多于文本框的宽度时，从中间省略文本。而且指定了 android:textAllCaps="true"，表明该文本框的所有字母大写。
- 第 3 个 TextView 指定 android:autoLink="email|phone"，这就指定了该文本框会自动为文本框内的 E-mail 地址、电话号码添加超链接。
- 第 4 个 TextView 指定系列 android:shadowXXX 属性，这就为该文本框内的文本内容添加了阴影。
- 第 5 个 TextView 指定 android:password="true"，这就指定了该文本框会用点来代替显示所有字符。
- 第 6 个 CheckedTextView 指定 android:checkMark="@drawable/ok"，这就指定了该可勾选文本框的勾选图标。



图 2.17　不同字体、颜色、带链接的文本

使用 Activity 来显示上面的布局文件，将看到如图 2.17 所示的界面。

实例：圆角边框、渐变背景的 TextView

默认情况下，TextView 是不带边框的，如果想为 TextView 添加边框，只能通过“曲线救国”的方式来实现——我们可以考虑为 TextView 设置一个背景 Drawable，该 Drawable 只是一个边框，这样就实现了带边框的 TextView。

由于可以为 TextView 设置背景 Drawable 对象，因此可以在定义 Drawable 时不仅指定边框，还可以指定渐变背景，这样即可为 TextView 添加渐变背景和边框。

下面的界面布局文件中定义了两个 TextView，界面布局文件的代码如下。

程序清单：codes\02\2.3\TextViewTest2\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 通过 android:background 指定背景 -->
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="带边框的文本"
      android:textSize="24pt"
      android:background="@drawable/bg_border"
      />
<!-- 通过 android:drawableLeft 绘制一张图片 -->
<TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="圆角边框、渐变背景的文本"
        android:textSize="24pt"
        android:background="@drawable/bg_border2"
        />
</LinearLayout>
```

上面的界面布局文件中定义了两个 TextView，其中第一个指定了背景，第二个定义文本框时指定使用圆角边框、渐变背景。第一个文本框所指定的背景是由 XML 文件定义的，将该文件放在 drawable_mdpi 文件夹内，该 XML 文件也可当成 Drawable 使用。下面是该 XML 文件的代码。

程序清单：codes\02\2.3\TextViewTest2\res\drawable-mdpi\bg_border.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置背景色为透明色 -->
      <solid android:color="#0000"/>
      <!-- 设置红色边框 -->
      <stroke android:width="4px" android:color="#f00" />
</shape>
```

第二个文本框所指定的背景是由 XML 文件定义的，将该文件放在 drawable_mdpi 文件夹内，该 XML 文件也可当成 Drawable 使用。下面是该 XML 文件的代码。

程序清单：codes\02\2.3\TextViewTest2\res\drawable-mdpi\bg_border2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <!-- 指定圆角矩形的 4 个圆角的半径 -->
      <corners android:topLeftRadius="20px"
    android:topRightRadius="5px"
    android:bottomRightRadius="20px"
    android:bottomLeftRadius="5px"/>
      <!-- 指定边框线条的宽度和颜色 -->
      <stroke android:width="4px" android:color="#f0f" />
      <!-- 指定使用渐变背景色，使用 sweep 类型的渐变
      颜色从红色→绿色→蓝色 -->
      <gradient android:startColor="#f00"
          android:centerColor="#0f0"
          android:endColor="#00f"
          android:type="sweep"/>
</shape>
```

使用 Activity 来显示上面定义的布局页面，可以看到如图 2.18 所示界面。

从图 2.18 不难看出，通过为 TextView 的 android:background 赋值，可以为文本框增加大量自定义外观，这种控制方式非常灵活。

需要指出的是，表面上这里只是在介绍 TextView，但由于 TextView 是 EditText、Button 等类的父类，因此此处介绍的对 TextView 控制的属性，同样适用于 EditText 与 Button。



图 2.18 圆角边框、渐变背景的文本

2.3.2 EditText 的功能与用法

EditText 与 TextView 非常相似，它甚至与 TextView 共用了绝大部分 XML 属性和方法。EditText 与 TextView 的最大区别在于：EditText 可以接受用户输入。表 2.13 中介绍的与输入相关的属性主要就是为 EditText 准备的。

EditText 组件最重要的属性是 inputType，该属性相当于 HTML 的<input.../>元素的 type 属性，用于 EditText 为指定类型的输入组件。inputType 能接受的属性值非常丰富，而且随着 Android 版本的升级，该属性能接受的类型还会增加。

EditText 还派生了如下两个子类。

- AutoCompleteTextView：带有自动完成功能的 EditText，实际上该组件的命名不太恰当。笔者认为该类名应该叫 AutoCompleteEditText 比较合适。由于该类通常需要与 Adapter 结合使用，因此将会在介绍 AdapteView 组件时介绍该组件的用法。
- ExtractEditText：它并不是 UI 组件，而是 EditText 组件的底层服务类，负责提供全屏输入法支持。

下面通过一个实例来介绍 EditText 的用法。

实例：用户友好的输入界面

对于一个用户友好的输入界面而言，接受用户输入的文本框内默认会提示用户如何输入；当用户把焦点切换到输入框时，输入框自动选中其中已输入的内容，避免用户删除已有内容；当用户把焦点切换到只接受电话号码的输入框时，输入法自动切换到数字键盘。

下面程序的输入界面完成了以上功能，输入界面的界面布局如下。

程序清单：codes\02\2.3\InputUI\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:stretchColumns="1"
      >
<TableRow>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="用户名:"
      android:textSize="16sp"
      />
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="请填写登录账号"
      android:selectAllOnFocus="true"
      />
</TableRow>
<TableRow>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="密码:"
      android:textSize="16sp"
```

```
        />
    <!-- android:inputType="numberPassword"表明只能接受数字密码 -->
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="numberPassword"
        />
    </TableRow>
    <TableRow>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="年龄："
        android:textSize="16sp"
        />
    <!-- inputType="number"表明是数值输入框 -->
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        />
    </TableRow>
    <TableRow>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="生日："
        android:textSize="16sp"
        />
    <!-- inputType="date"表明是日期输入框 -->
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="date"
        />
    </TableRow>
    <TableRow>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="电话号码:"
        android:textSize="16sp"
        />
    <!-- inputType="phone"表明是输入电话号码的输入框 -->
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="请填写您的电话号码"
        android:selectAllOnFocus="true"
        android:inputType="phone"
        />
    </TableRow>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="注册"
        />
    </TableLayout>
```

上面的界面布局中第一个文本框通过 android:hint 指定了文本框的提示信息：请填写登录账号——这是该文本框默认的提示。当用户还没有输入时，该文本框内默认显示这段信息：



图 2.19　友好的输入界面

第二个输入框通过android:inputType="numberPassword"设置这是一个密码框，而且只能接受数字密码，用户在该文本框输入的字符会以点号代替；第三个输入框通过 android: inputType="number"设置为只能接受数值的输入框；第四个输入框通过 android:inputType= "date"指定它是一个日期选输入框；第五个输入框通过 android:inputType= "phone"设置为一个电话号码输入框。

使用 Activity 显示上面的界面布局将可看到如图 2.19 所示的界面。

从图 2.19 可以看出，当用户把焦点定位到数字密码输入框时，系统自动显示数字输入键盘，这就是设置 android:inputType= "numberPassword"的作用。第一个编辑框默认显示了“请填写登录账号”，这是由 android:hint 属性指定的。

2.3.3　按钮（Button）组件的功能和用法

Button 继承了 TextView，它主要是在 UI 界面上生成一个按钮，该按钮可以供用户单击，当用户单击按钮时，按钮会触发一个 onClick 事件。关于 onClick 事件编程的简单示例，本书前面已经见到很多，后面还会详细介绍 Android 的事件编程。

按钮使用起来比较容易，可以通过为按钮指定 android:background 属性为按钮增加背景颜色或背景图片，如果将背景图片设为不规则的背景图片，则可以开发出各种不规则形状的按钮。

如果只是使用普通的背景颜色或背景图片，那么这些背景是固定的，不会随着用户的动作而改变。如果需要让按钮的背景颜色、背景图片随用户动作动态改变，则可以考虑使用自定义 Drawable 对象来实现。

下面通过实例来开发出更强大的按钮。

实例：按钮、圆形按钮、带文字的图片按钮

为了定义图片随用户动作改变的按钮，可以考虑使用 XML 资源文件来定义 Drawable 对象，再将 Drawable 对象设为 Button 的 android:background 属性值，或设为 ImageButton 的 android:src 属性值。

例如有如下界面布局文件。

程序清单：codes\02\2.3\ButtonTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 文字带阴影的按钮 -->
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="文字带阴影的按钮"
      android:textSize="12pt"
      android:shadowColor="#aa5"
      android:shadowRadius="1"
```

```
        android:shadowDx="5"
        android:shadowDy="5"
    />
    <!-- 普通文字按钮 -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/red"
        android:text="普通按钮"
        android:textSize="10pt"
    />
    <!-- 带文字的图片按钮-->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/button_selector"
        android:textSize="11px"
        android:text="带文字的图片按钮"
    />
    </LinearLayout>
```

上面的界面布局中第一个按钮是一个普通按钮，但为该按钮的文字指定了阴影——配置阴影的方式与为 TextView 配置阴影的方式完全相同，这是因为 Button 的本质还是 TextView。第二个按钮通过 background 属性配置了背景图片，因此该按钮将会显示为背景图片形状的按钮。

第三个按钮有点特殊，它指定了 android:background 属性为@drawable/button_selector，该属性值引用一个 Drawable 资源，该资源对应的 XML 文件如下。

程序清单：codes\02\2.3\ButtonTest\res\drawable-mdpi\button_selector.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 指定按钮按下时的图片 -->
    <item android:state_pressed="true"
        android:drawable="@drawable/red"
    />
    <!-- 指定按钮松开时的图片 -->
    <item android:state_pressed="false"
        android:drawable="@drawable/purple"
    />
</selector>
```

上面的资源文件使用<selector.../>元素定义了一个 StateListDrawable 对象——本书后面会详细介绍如何使用 XML 文件来定义 Drawable 的内容，此处不再深入讲解。

使用 Activity 显示上面的布局文件，可以看到如图 2.20 所示的界面。



图 2.20 各种按钮

在图 2.20 所示界面的三个按钮中，前两个按钮的背景色、图片都是固定的，用户单击该按钮不会看到任何改变；用户按下后面两个按钮时，将会看到按钮的图片被切换为红色。

以笔者的经验来看，Button 生成的按钮功能很强大，就像第三个按钮就是 Button 生成的，而且它也可以通过背景色来设置图片，因此使用 Button 生成的按钮不仅可以是普通的文字按钮，也可以定制成任意形状，并可以随用户交互动作改变外观。

2.3.4 使用 9Patch 图片作为按钮背景

从图 2.20 的第三个按钮来看，当按钮的内容太多时，Android 会自动缩放整张图片，以保证背景图片能覆盖整个按钮。但这种缩放整张图片的效果可能并不好。可能需要的情况是我们只想缩放图片中某个部分，这样才能保证按钮的视觉效果。

为了实现只缩放图片中某个部分的效果，我们需要借助于 9Patch 图片来实现。9Patch 图片是一种特殊的 PNG 图片，这种图片以.9.png 结尾，它在原始图片四周各添加一个宽度为 1 像素的线条，这 4 条线就决定了该图片的缩放规则、内容显示规则。

左侧和上侧的直线共同决定了图片的缩放区域：以左边直线为左边界绘制矩形，它覆盖的区域可以在纵向缩放；以上面直线为上边界绘制矩形，它覆盖的区域可以水平缩放；它们二者的交集可以在两个方向上缩放。图 2.21 显示了定义图片缩放区域的示意。

右侧和下侧的直线共同决定图片的内容显示区域：以右边直线为右边界绘制矩形，以下边直线为下边界绘制矩形，它们二者的交集就是图片的内容显示区域。图 2.22 显示了定义图片内容显示区域的示意。



图 2.21 定义图片缩放区域

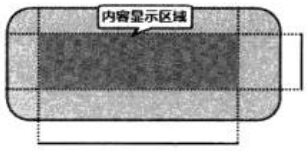


图 2.22 定义图片的内容显示区

Android 为制作 9Patch 图片提供了 draw9patch 工具，该工具位于 Android SDK 安装路径的 tools 目录下，进入该目录双击 draw9patch.bat 文件，即可启动该工具。



提示：

如果启动该程序时提示 java.lang.NoClassDefFoundError: org/jdesktop/swingworker/SwingWorker 异常。可能的原因是 draw9patch.bat 工具需要依赖于 SwingWorker 类，这个类在 JDK 1.5 以前需要单独下载 swing-worker 的 JAR 包，但从 JDK 1.6 以后，该类已经加入了 javax.swing 包下，但 Android SDK 还在使用早期的 SwingWorker，而且 Android SDK 又删除了 swing-worker-1.1.jar 包，所以引发上面的错误。为了能正常启动 draw9patch.bat，手动将 swing-worker-1.1.jar 复制到 Android SDK 安装路径的 tools/lib 路径下即可。

启动 draw9patch.bat 之后，通过该工具主菜单上的“File→Open 9 - Patch”菜单项打开一张 PNG 图片，然后通过该工具定义图片的缩放区域、内容显示区域。图 2.23 显示了定义 9Patch 图片的示意。

将上面生成的图片保存到应用的 res/drawable-mdpi 路径下，主文件名任意，draw9patch.bat 自动将该文件的后缀保存为.9.png。

定义 9Patch 图片之后，接下来在应用中定义两个按钮，分别使用原始图片和 9Patch 图片作为 Button 的背景。页面定义文件比较简单，此处不再赘述。普通图片为背景的按钮及 9Patch 图片为背景的按钮的界面如图 2.24 所示。

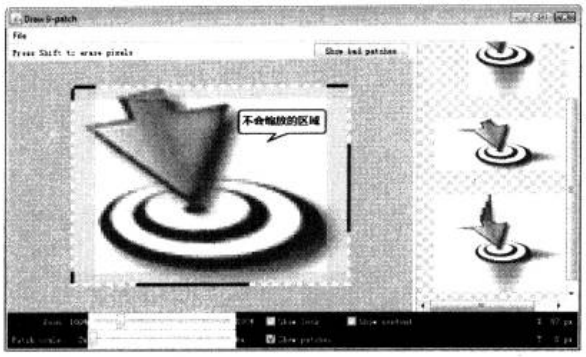


图 2.23 定义 9Patch 图片

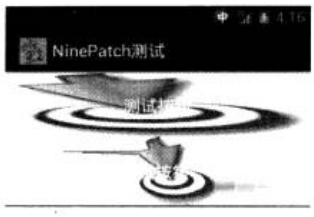


图 2.24 普通图片与 9Patch 图片作为背景的对比

从图 2.24 可以看出，普通图片作为背景时，整张图片都被缩放了，如果使用 9Patch 图片作为背景，则只有指定区域才会被缩放。

2.3.5 单选按钮（RadioButton）与复选框（CheckBox）的功能与用法

单选按钮（RadioButton）和复选框（CheckBox）、状态开关按钮（ToggleButton）与开关（Switch）是用户界面中最普通的 UI 组件，它们都继承了 Button 类，因此都可直接使用 Button 支持的各种属性和方法。

RadioButton、CheckBox 与普通按钮不同的是，它们多了一个可选中的功能，因此 RadioButton、CheckBox 都可额外指定一个 android:checked 属性，该属性用于指定 RadioButton、CheckBox 初始时是否被选中。

RadioButton 与 CheckBox 的不同之处在于，一组 RadioButton 只能选中其中一个，因此 RadioButton 通常要与 RadioGroup 一起使用，用于定义一组单选按钮。

下面通过实例来介绍 RadioButton 和 CheckBox 的用法。

实例：利用单选按钮、复选框获取用户信息

在需要获取用户信息的界面中，有些信息不需要用户直接输入，可以考虑让用户进行选择，比如用户的性别、爱好等。下面的界面布局文件定义了一个让用户选择的输入界面。

程序清单：codes\02\2.3\CheckButtonTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<TableRow>
<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="性别:"
      android:textSize="16px"
      />
<!-- 定义一组单选按钮 -->
<RadioGroup android:id="@+id/rg"
      android:orientation="horizontal"
```

```
        android:layout_gravity="center_horizontal">
    <!-- 定义两个单选按钮 -->
    <RadioButton android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/male"
        android:text="男"
        android:checked="true"
        />
    <RadioButton android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/female"
        android:text="女"
        />
    </RadioGroup>
    </TableRow>
    <TableRow>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="喜欢的颜色:"
        android:textSize="16px"
        />
    <!-- 定义一个垂直的线性布局 -->
    <LinearLayout android:layout_gravity="center_horizontal"
        android:orientation="vertical"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    <!-- 定义三个复选框 -->
    <CheckBox android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="红色"
        android:checked="true"
    />
    <CheckBox android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="蓝色"
    />
    <CheckBox android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="绿色"
    />
    </LinearLayout>
    </TableRow>
    <TextView
        android:id="@+id/show"
       android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    </TableLayout>
```

上面的界面布局中定义了一组单选按钮，并默认勾选了第一个单选按钮，这组单选按钮供用户选择性别；还定义了三个复选框，供用户选择喜欢的颜色。

注意：

如果在 XML 布局文件中默认勾选了某个单选按钮，则必须为该组单选按钮的每个按钮指定 android:id 属性值，否则这组单选按钮不能正常工作。



为了监听单选按钮、复选框的勾选状态的改变，可以为它们添加事件监听器。例如下面

Activity 为 RadioGroup 添加了事件监听器，该监听器可以监听这组单选按钮的勾选状态的改变。

程序清单：codes\02\2.3\CheckButtonTest\src\org\crazyit\ui\CheckButtonTest.java

```
public class CheckButtonTest extends Activity
{
    RadioGroup rg;
    TextView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面上 rg、show 两个组件
        rg = (RadioGroup) findViewById(R.id.rg);
        show = (TextView) findViewById(R.id.show);
        // 为 RadioGroup 组件的 OnCheck 事件绑定事件监听器
        rg.setOnCheckedChangeListener(new OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId)
            {
                // 根据用户勾选的单选按钮来动态改变 tip 字符串的值
                String tip = checkedId == R.id.male ?
                        "您的性别是男人": "您的性别是女人";
                // 修改 show 组件中的文本
                show.setText(tip);
            }
        });
    }
}
```



提示：上面代码添加事件监听器的方式采用了“委托式”事件处理机制，委托式事件处理机制的原理是：当事件源上发生事件时，该事件将会激发该事件源上的监听器的特定方法。本书第 3 章还会详细介绍这种事件处理机制。

运行上面的程序，并改变第一组单选按钮的勾选状态，将看到如图 2.25 所示界面。



图 2.25 单选按钮、复选框示意

2.3.6 状态开关按钮（ToggleButton）与开关（Switch）的功能与用法

状态开关按钮（ToggleButton）与开关（Switch）也是由 Button 派生出来的，因此它们的本质也是按钮，Button 支持的各种属性、方法也适用于 ToggleButton 和 Switch。从功能上来看，ToggleButton、Switch 与 CheckBox 复选框非常相似，它们都可以提供两个状态。不过 ToggleButton、Switch 与 CheckBox 的区别主要体现在功能上，ToggleButton、Switch 通常用于切换程序中的某种状态。

表 2.14 显示了 ToggleButton 所支持的 XML 属性及相关方法的说明。

表 2.14　ToggleButton 支持的 XML 属性及相关方法说明

XML 属性	相关方法	说　明
android:checked	setChecked(boolean)	设置该按钮是否被选中
android:textOff		设置当该按钮的状态关闭时显示的文本
android:textOn		设置当该按钮的状态打开时显示的文本

表 2.15 显示了 Switch 所支持的 XML 属性及相关方法的说明。

表 2.15　Switch 支持的 XML 属性及相关方法说明

XML 属性	相关方法	说　明
android:checked	setChecked(boolean)	设置该开关是否被选中
android:switchMinWidth	setSwitchMinWidth(int)	设置该开关的最小宽度
android:switchPadding	setSwitchPadding(int)	设置开关与标题文本之间的空白
android:switchTextAppearance	setSwitchTextAppearance(Context,int)	设置该开关图标上的文本样式
android:textOff	setTextOff(CharSequence)	设置该开关的状态关闭时显示的文本
android:textOn	setTextOn(CharSequence)	设置该开关的状态打开时显示的文本
android:textStyle	setSwitchTypeface(Typeface)	设置该开关的文本的风格
android:thumb	setThumbResource(int)	指定使用自定义 Drawable 绘制该开关的开关按钮
android:track	setTrackResource(int)	指定使用自定义 Drawable 绘制该开关的开关轨道
android:typeface	setSwitchTypeface(Typeface)	设置该开关的文本的字体风格

下面通过一个动态控制布局的实例来示范 ToggleButton 与 Switch 的用法。

实例：动态控制布局

该实例的思路是在页面中增加一个 ToggleButton，随着该按钮状态的改变，界面布局中的 LinearLayout 布局的方向在水平布局、垂直布局之间切换。下面是该程序所使用的界面布局。

程序清单：codes\02\2.3\ToggleButtonTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 定义一个 ToggleButton 按钮 -->
<ToggleButton android:id="@+id/toggle"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textOff="横向排列"
      android:textOn="纵向排列"
      android:checked="true"
      />
<Switch android:id="@+id/switcher"
    android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textOff="横向排列"
      android:textOn="纵向排列"
      android:thumb="@drawable/check"
      android:checked="true"/>
<!-- 下面省略了三个按钮的定义 -->
```

```
...
</LinearLayout>
</LinearLayout>
```

上面 LinearLayout 中定义了三个按钮，该 LinearLayout 默认采用垂直方向的线性布局。接下来我们为 ToggleButton 按钮、Switch 按钮绑定监听器，当它的选中状态发生改变时，程序通过代码来改变 LinearLayout 的布局方向。

程序清单：codes\02\2.3\ToggleButtonTest\src\org\crazyit\ui\ToggleButtonTest.java

```
public class ToggleButtonTest extends Activity
{
    ToggleButton toggle;
    Switch switcher;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        toggle = (ToggleButton)findViewById(R.id.toggle);
        switcher = (Switch)findViewById(R.id.switcher);
        final LinearLayout test = (LinearLayout)findViewById(R.id.test);
        OnCheckedChangeListener listener = new OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(CompoundButton button
                    , boolean isChecked)
            {
                if(isChecked)
                {
                    //设置 LinearLayout 垂直布局
                    test.setOrientation(1);
                }
                else
                {
                    //设置 LinearLayout 水平布局
                    test.setOrientation(0);
                }
            }
        };
        toggle.setOnCheckedChangeListener(listener);
        switcher.setOnCheckedChangeListener(listener);
    }
}
```

运行上面的程序，随着用户改变 ToggleButton 按钮的状态，下面界面布局的方向也在不断发生改变。图 2.26 显示了 ToggleButton 的界面。



图 2.26 ToggleButton 与 Switch 的功能

➤➤2.3.7 时钟（AnalogClock 和 DigitalClock）的功能与用法

时钟 UI 组件是两个非常简单的组件，DigitalClock 本身就继承了 TextView——也就是说它本身就是文本框，只是它里面显示的内容总是当前时间。与 TextView 不同的是，为 DigitalClock 设置 android:text 属性没什么作用。

AnalogClock 则继承了 View 组件，它重写了 View 的 OnDraw 方法，它会在 View 上绘制

模拟时钟。

表 2.16 显示了 AnalogClock 所支持的 XML 属性的说明。

表 2.16　AnalogClock 支持的 XML 属性的说明

XML 属性	说　明
android:dial	设置该模拟时钟的表盘使用的图片
android:hand_hour	设置该模拟时钟的时针表盘使用的图片
android:hand_minute	设置该模拟时钟的分针使用的图片

DigitalClock 和 AnalogClock 都会显示当前时间。不同的是，DigitalClock 显示数字时钟，可以显示当前的秒数；AnalogClock 显示模拟时钟，不会显示当前秒数。

下面通过实例来示范 AnalogClock 和 DigitalClock 的用法。

实例：手机里的“劳力士”

由于我们可以通过图片定制 AnalogClock 模拟指针的表盘、时针、分针，因此只要使用合适的图片，就可以对 AnalogClock 进行任意定制。下面的实例将会使用“劳力士”图片来定义模拟时钟，从而开发手机里的“劳力士”。

下面是本实例的布局文件。

程序清单：codes\02\2.3\ClockTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<!-- 定义模拟时钟 -->
<AnalogClock
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      />
<!-- 定义数字时钟 -->
<DigitalClock
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="14pt"
      android:textColor="#f0f"
      android:drawableRight="@drawable/ic_launcher"
      />
<!-- 定义模拟时钟，并使用自定义表盘、时针图片 -->
<AnalogClock
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:dial="@drawable/watch"
      android:hand_minute="@drawable/hand"
      />
</LinearLayout>
```

使用 Activity 显示上面的界面布局，将看到如图 2.27 所示的界面。

正如从上面的粗体字代码中看到的，如果想控制模拟时钟显示时间的字号大小、字体颜色等，都可通过 android:textSize、android:textColor 等属性进行控制——因为 DigitalClock 的

本质还是一个 TextView，所以它可以使用 TextView 的 XML 属性和方法。



图 2.27 数字时钟与模拟时钟

2.3.8 计时器（Chronometer）

Android 还提供了一个计时器组件：Chronometer，该组件与 DigitalClock 都继承自 TextView，因此它们都会显示一段文本。但 Chronometer 并不显示当前时间，它显示的是从某个起始时间开始，一共过去了多长时间。

Chronometer 的用法也很简单，它只提供了一个 android:format 属性，用于指定计时器的计时格式。除此之外，Chronometer 支持如下常用方法。

- setBase(long base)：设置计时器的起始时间。
- setFormat(String format)：设置显示时间的格式。
- start()：开始计时。
- stop()：停止计时。
- setOnChronometerTickListener(Chronometer.OnChronometerTickListener listener)：为计时器绑定事件监听器，当计时器改变时触发该监听器。

下面的程序简单示范了 Chronometer 的用法，该程序界面中定义了一个 Chronometer 组件和一个 Button 组件。当用户单击 Button 时系统开始计时，当计时超过 20 秒时停止计时。下面是该程序的代码。

程序清单：codes\02\2.3\ChronometerTest\src\org\crazyit\time\ChronometerTest.java

```
public class ChronometerTest extends Activity
{
    Chronometer ch;
    Button start;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取计时器组件
        ch = (Chronometer) findViewById(R.id.test);
        // 获取“开始”按钮
        start = (Button) findViewById(R.id.start);
        start.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 设置开始计时时间
                ch.setBase(SystemClock.elapsedRealtime());
                // 启动计时器
                ch.start();
                start.setEnabled(false);
            }
        });
        // 为 Chronometer 绑定事件监听器
        ch.setOnChronometerTickListener(new OnChronometerTickListener()
        {
```

```
                @Override
                public void onChronometerTick(Chronometer ch)
                {
                    // 如果从开始计时到现在超过了20s
                    if (SystemClock.elapsedRealtime() - ch.getBase() > 20 * 1000)
                    {
                        ch.stop();
                        start.setEnabled(true);
                    }
                }
            });
        }
    }
```

上面的程序中粗体字代码用于设置 Chronometer 的起始时间，并启动 Chronometer。启动计时器后可以看到该组件显示已流逝的时间，如图 2.28 所示。



图 2.28　计时器

程序中用到的 SystemClock 类仅仅是一个获取系统时间、运行时间的工具类，用法很简单，读者自行查阅 API 文档即可。

2.4　第 3 组 UI 组件：ImageView 及其子类

ImageView 继承自 View 组件，它的主要功能是用于显示图片——实际上这个说法不太严谨，因为它能显示的不仅仅是图片，任何 Drawable 对象都可使用 ImageView 来显示。

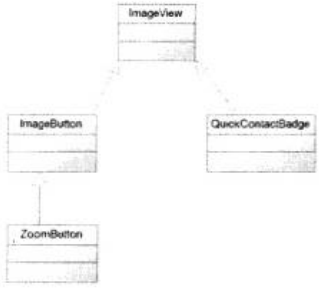


图 2.29　ImageView 及其子类的类继承图

除此之外，ImageView 还派生了 ImageButton、ZoomButton 等组件，图 2.29 显示了 ImageView 及其子类的类图。

从图 2.29 可以看出，ImageView 派生了 ImageButton、ZoomButton 等组件，因此 ImageView 支持的 XML 属性、方法，基本上也可应用于 ImageButton、ZoomButton 等组件。

表 2.17 显示了 ImageView 支持的常用 XML 属性及相关方法的说明。

表 2.17　ImageView 支持的 XML 属性及相关方法的说明

XML 属性	相关方法	说　明
android:adjustViewBounds	setAdjustViewBounds(boolean)	设置 ImageView 是否调整自己的边界来保持所显示图片的长宽比
android:cropToPadding	setCropToPadding(boolean)	如果将该属性设为 true，该组件将会被裁剪到保留该 ImageView 的 padding
android:maxHeight	setMaxHeight(int)	设置 ImageView 的最大高度
android:maxWidth	setMaxWidth(int)	设置 ImageView 的最大宽度
android:scaleType	setScaleType(ImageView.ScaleType)	设置所显示的图片如何缩放或移动以适应 ImageView 的大小
android:src	setImageResource(int)	设置 ImageView 所显示的 Drawable 对象的 ID

表 2.17 所支持的 android:scaleType 属性可指定如下属性值。

- matrix（ImageView.ScaleType.MATRIX）：使用 matrix 方式进行缩放。
- fitXY（ImageView.ScaleType.FIT_XY）：对图片横向、纵向独立缩放，使得该图片完全适应于该 ImageView，图片的纵横比可能会改变。
- fitStart（ImageView.ScaleType.FIT_START）：保持纵横比缩放图片，直到该图片能完全显示在 ImageView 中（图片较长的边长与 ImageView 相应的边长相等），缩放完成后将该图片放在 ImageView 的左上角。
- fitCenter（ImageView.ScaleType.FIT_CENTER）：保持纵横比缩放图片，直到该图片能完全显示在 ImageView 中（图片较长的边长与 ImageView 相应的边长相等），缩放完成后将该图片放在 ImageView 的中央。
- fitEnd（ImageView.ScaleType.FIT_END）：保持纵横比缩放图片，直到该图片能完全显示在 ImageView 中（图片较长的边长与 ImageView 相应的边长相等），缩放完成后将该图片放在 ImageView 的右下角。
- center（ImageView.ScaleType.CENTER）：把图片放在 ImageView 的中间，但不进行任何缩放。
- centerCrop（ImageView.ScaleType.CENTER_CROP）：保持纵横比缩放图片，以使得图片能完全覆盖 ImageView。只要图片的最短边能显示出来即可。
- centerInside（ImageView.ScaleType.CENTER_INSIDE）：保持纵横比缩放图片，以使得 ImageView 能完全显示该图片。

为了控制 ImageView 显示的图片，ImageView 提供了如下方法。

- setImageBitmap(Bitmap bm)：使用 Bitmap 位图设置该 ImageView 显示的图片。
- setImageDrawable(Drawable drawable)：使用 Drawable 对象设置该 ImageView 显示的图片。
- setImageResource(int resId)：使用图片资源 ID 设置该 ImageView 显示的图片。
- setImageURI(Uri uri)：使用图片的 URI 设置该 ImageView 显示的图片。

ImageView 的功能比较简单，下面结合一个图片浏览器的实例来示范 ImageView 的功能和用法。

实例：图片浏览器

本例的图片浏览器可以改变所查看图片的透明度，可通过调用 ImageView 的 setAlpha 方法来实现。不仅如此，本图片浏览器还可通过一个小区域来查看图片的原始大小，因此本例会定义两个 ImageView，一个用于查看图片整体，一个用于查看图片局部的细节。

下面是本例的界面布局文件。

程序清单：codes\02\2.4\ImageViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center">
```

```
<!-- 此处省略三个按钮定义 -->
...
</LinearLayout>
<!-- 定义显示图片整体的 ImageView -->
<ImageView android:id="@+id/image1"
      android:layout_width="fill_parent"
      android:layout_height="240px"
      android:src="@drawable/shuangta"
      android:scaleType="fitCenter"/>
<!-- 定义显示图片局部细节的 ImageView -->
<ImageView android:id="@+id/image2"
      android:layout_width="120dp"
      android:layout_height="120dp"
      android:background="#00f"
      android:layout_marginTop="10dp"/>
</LinearLayout>
```

上面的界面布局中定义了两个 ImageView，其中第一个 ImageView 指定了 android:scaleType= "fitCenter"，这表明 ImageView 显示图片时会进行保持纵横比的缩放，并将缩放后的图片放在该 ImageView 的中央。

为了能动态改变图片的透明度，接下来需要为按钮编写事件监听器，当用户单击按钮时动态改变图片的 Alpha 值。为了能动态显示图片的局部细节，程序为第一个 ImageView 添加 OnTouchListener 监听器，用户在第一个 ImageView 上发生触摸事件时，程序从原始图片中读取相应部分的图片，并将其显示在第二个 ImageView 中。下面是主程序的代码。

程序清单：codes\02\2.4\ImageViewTest\src\org\crazyit\ui\ImageViewTest.java

```
public class ImageViewTest extends Activity
{
      // 定义一个访问图片的数组
      int[] images = new int[]{
            R.drawable.lijiang,
            R.drawable.qiao,
            R.drawable.shuangta,
            R.drawable.shui,
            R.drawable.xiangbi,
      };
      // 定义默认显示的图片
      int currentImg = 2;
      // 定义图片的初始透明度
      private int alpha = 255;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            final Button plus = (Button) findViewById(R.id.plus);
            final Button minus = (Button) findViewById(R.id.minus);
            final ImageView image1 = (ImageView) findViewById(R.id.image1);
            final ImageView image2 = (ImageView) findViewById(R.id.image2);
            final Button next = (Button) findViewById(R.id.next);
            // 定义查看下一张图片的监听器
            next.setOnClickListener(new OnClickListener()
            {
                  @Override
                  public void onClick(View v)
                  {
```

```
            // 控制 ImageView 显示下一张图片
            image1.setImageResource(
                  images[++currentImg % images.length]);
      }
});
// 定义改变图片透明度的方法
OnClickListener listener = new OnClickListener()
{
      @Override
      public void onClick(View v)
      {
            if (v == plus)
            {
                  alpha += 20;
            }
            if (v == minus)
            {
                  alpha -= 20;
            }
            if (alpha >= 255)
            {
                  alpha = 255;
            }
            if (alpha <= 0)
            {
                  alpha = 0;
            }
            // 改变图片的透明度
            image1.setAlpha(alpha);
      }
};
// 为两个按钮添加监听器
plus.setOnClickListener(listener);
minus.setOnClickListener(listener);
image1.setOnTouchListener(new OnTouchListener()
{
      @Override
      public boolean onTouch(View view, MotionEvent event)
      {
            BitmapDrawable bitmapDrawable = (BitmapDrawable) image1
                        .getDrawable();
            // 获取第一个图片显示框中的位图
            Bitmap bitmap = bitmapDrawable.getBitmap();
            // bitmap 图片实际大小与第一个 ImageView 的缩放比例
            double scale = bitmap.getWidth() / 320.0;
            // 获取需要显示的图片的开始点
            int x = (int) (event.getX() * scale);
            int y = (int) (event.getY() * scale);
            if (x + 120 > bitmap.getWidth())
            {
                  x = bitmap.getWidth() - 120;
            }
            if (y + 120 > bitmap.getHeight())
            {
                  y = bitmap.getHeight() - 120;
            }
            // 显示图片的指定区域
            image2.setImageBitmap(Bitmap.createBitmap(bitmap
```

```
                                    , x, y, 120, 120));
                            image2.setAlpha(alpha);
                            return false;
                        }
                    });
            }
    }
```

上面的程序中第一行粗体字代码调用了 ImageView 的 setImageResoure()方法动态修改该 ImageView 所显示的图片，当用户单击“下一张”按钮时，即可控制 ImageView 显示图片数组中的下一张图片。程序第二行粗体字代码会动态设置第一个 ImageView 的 Alpha 值，这就是动态改变该图片的透明度了。

程序中的粗体字代码段将会用于从源位图的触碰点“截取”源位图，并将截取的部分显示在第二个 ImageView 中。此时用到了 Bitmap 类，它是一个代表位图的类，调用它的 createBitmap()静态方法即可截取位图的指定部分，该方法返回截取区域生成的新位图。



图 2.30　使用 ImageView 实现图片浏览器

运行上面的程序，将看到如图 2.30 所示的界面。

ImageView 派生了如下两个子类。

- ImageButton：图片按钮。
- QuickContactBadge：显示关联到特定联系人的图片。

Button 与 ImageButton 的区别在于，Button 生成的按钮上显示文字，而 ImageButton 上则显示图片。需要指出的是：为 ImageButton 按钮指定 android:text 属性没用（ImageButton 的本质是 ImageView），即使指定了该属性，图片按钮上也不会显示任何文字。

如果考虑使用 ImageButton，图片按钮可以指定 android:src 属性，该属性既可使用静止的图片，也可使用自定义的 Drawable 对象，这样即可开发出随用户动作改变图片的按钮。

ImageButton 派生了一个 ZoomButton，ZoomButton 可以代表“放大”、“缩小”两个按钮。ZoomButton 的行为基本类似于 ImageButton，只是 Android 默认提供了 btn_minus、btn_plus 两个 Drawable 资源，只要为 ZoomButton 的 android:src 属性分别指定 btn_minus、btn_plus，即可实现“放大”、“缩小”按钮。



提示：
实际上 Android 还提供了一个 ZoomControls 组件，该组件相当于同时组合了“放大”、“缩小”两个按钮，并允许分别为两个按钮绑定不同的事件监听器。

实例：强大的图片按钮

下面的实例定义了多个图片按钮，并定义了两个 ZoomButton。两个 ZoomButton 的 android:src 属性分别指定为@android:drawable/btn_minus、@android:drawable/btn_plus，这样即可定义“缩小”和“放大”两个按钮。

下面是该实例的界面布局文件。

程序清单：codes\02\2.4\ImageButtonTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
    <!-- 普通图片按钮 -->
    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/blue"
    />
    <!-- 按下时显示不同图片的按钮 -->
    <ImageButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/button_selector"
    />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10sp"
        android:layout_gravity="center_horizontal">
        <!-- 分别定义 2 个 ZoomButton，并分别似乎用 btn_minus 和 btn_plus 图片 -->
        <ZoomButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/btn_zoom_down"
            android:src="@android:drawable/btn_minus" />
        <ZoomButton
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/btn_zoom_up"
            android:src="@android:drawable/btn_plus" />
    </LinearLayout>
    <!-- 定义 ZoomControls 组件 -->
    <ZoomControls android:id="@+id/zoomControls1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"/>
    </LinearLayout>
```

上面布局文件的开头定义了两个 ImageButton，第一个 ImageButton 的 android:src 指定为一张静态图片，这样无论用户有怎样的行为，该 ImageButton 总显示这张静态图片。第二个 ImageButton 的 android:src 指定为@drawable/button_selector，该 Drawable 组合了两张图片，可以保证用户单击该按钮时切换图片。

该布局文件中间定义了两个 ZoomButton，并分别指定了放大、缩小 Drawable，该布局文件的结尾处定义了一个 ZoomControls 组件。使用 Activity 显示该布局文件，可以看到如图 2.31 所示效果。

对于第二个图片按钮，当用户按下第二个图片按钮时，将会看到按钮的图片被切换为红色图片。



图 2.31 强大的图片按钮

实例：使用 QuickContactBadge 关联联系人

QuickContactBadge 继承了 ImageView，因此它的本质也是图片，也可以通过 android:src 属性指定它显示的图片。QuickContactBadge 额外增加的功能是：该图片可以关联到手机中指

定联系人，当用户单击该图片时，系统将会打开相应联系人的联系方式界面。

为了让 QuickContactBadge 与特定联系人关联，可以调用如下方法进行关联。

- assignContactFromEmail(String emailAddress, boolean lazyLookup)：将该图片关联到指定 E-mail 地址对应的联系人。
- assignContactFromPhone(String phoneNumber, boolean lazyLookup)：将该图片关联到指定电话号码对应的联系人。
- assignContactUri(Uri contactUri)：将该图片关联到特定 Uri 对应的联系人。

下面的实例示范了如何使用 QuickContactBadge 关联特定联系人。

该实例的界面布局文件如下。

程序清单：codes\02\2.4\QuickContactBadgeTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<QuickContactBadge
      android:id="@+id/badge"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:src="@drawable/ic_launcher"/>
<TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="16dp"
      android:text="我的偶像"/>
</LinearLayout>
```

上面的布局文件非常简单，它只是包含了一个 QuickContactBadge 组件与 TextView 组件。接下来在 Activity 代码中可以让 QuickContactBadge 与特定联系人建立关联。下面是该 Activity 的代码。

程序清单：codes\02\2.4\QuickContactBadgeTest\src\org\crazyit\ui\QuickContactBadgeTest.java

```
public class QuickContactBadgeTest extends Activity
{
    QuickContactBadge badge;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 QuickContactBadge 组件
        badge = (QuickContactBadge) findViewById(R.id.badge);
        // 将 QuickContactBadge 组件与特定电话号码对应的联系人建立关联
        badge.assignContactFromPhone("020-88888888", false);
    }
}
```

上面的粗体字代码将该 QuickContactBadge 组件与电话号码为 02088888888 的联系人建立了关联，当用户单击该图片时，系统将会打开该联系人对应的联系方式界面。

运行该实例，可以看到如图 2.32 所示界面。

图 2.32 左边的图片就是 QuickContactBadge 组件，单击该组件，如果手机中存储有电话号码为 02088888888 的联系人，系统将会打开该联系人的联系方式界面，如图 2.33 所示。



图 2.32 使用 QuickContactBadge



图 2.33 使用 QuickContactBadge 打开特定联系人

2.5 第 4 组 UI 组件：AdapterView 及子类

AdapterView 组件是一组重要的组件，AdapterView 本身是一个抽象基类，它派生的子类在用法上十分相似，只是显示界面有一定的区别，因此本节把它们归为一类，针对它们的共性集中讲解，并突出介绍它们的区别。

AdapterView 具有如下特征。

- AdapterView 继承了 ViewGroup，它的本质是容器。
- AdapterView 可以包括多个“列表项”，并将多个“列表项”以合适的形式显示出来。
- AdapterView 显示的多个“列表项”由 Adapter 提供。调用 AdapterView 的 setAdapter(Adapter)方法设置 Adapter 即可。

AdapterView 及其子类的继承关系类图如图 2.34 所示。

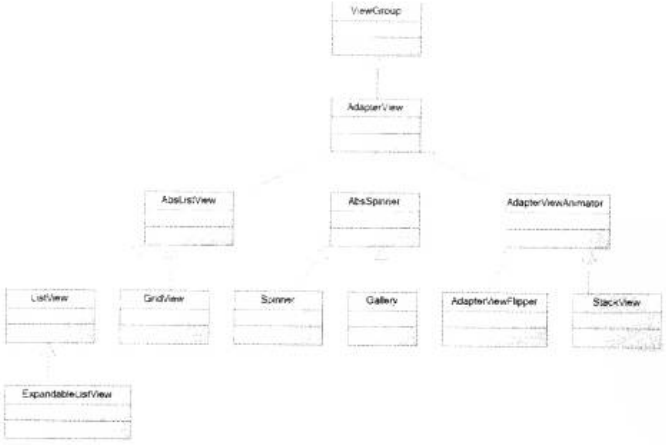


图 2.34 AdapterView 及其子类的继承关系图

从图 2.34 不难看出，AdapterView 派生了三个子类：AbsListView、AbsSpinner 和 AdapterViewAnimator，这三个子类依然是抽象的，实际使用时往往采用它们的子类。

2.5.1 列表视图（ListView）和 ListActivity

ListView 是手机系统中使用非常广泛的一种组件，它以垂直列表的形式显示所有列表项。

创建 ListView 有如下两种方式。

- 直接使用 ListView 进行创建。
- 让 Activity 继承 ListActivity（相当于该 Activity 显示的组件为 ListView）。

一旦在程序中获得了 ListView 之后，接下来就需要为 ListView 设置它要显示的列表项了。在这一点上，ListView 表显示出 AdapterView 的特征：通过 setAdapter(Adapte)方法为之提供 Adapter、并由 Adapter 提供列表项即可。



提示：

ListView、GridView、Spinner、Gallery 等 AdapterView 都只是容器，而 Adapter 负责提供每个“列表项”组件，AdapterView 则负责采用合适的方式显示这些列表项。

AbsListView 提供了如表 2.18 所示的 XML 属性。

表 2.18　AbsListView 的常用 XML 属性

XML 属性	相关方法	说　明
android:choiceMode		设置 AbsListView 的选择行为。该属性支持如下属性值。 ➢ none：不显示任何选中项 ➢ singleChoice：允许单选 ➢ multipleChoice：允许多选 ➢ multipleChoiceModal：允许多选
android:drawSelectorOnTop	setDrawSelectorOnTop(boolean)	如果该属性设为 true，选中的列表项将会显示在上面
android:fastScrollEnabled		设置是否允许快速滚动。如果该属性设为 true，将会显示滚动图标，并允许用户拖动该滚动图标进行快速滚动
android:listSelector	setSelector(int)	指定被选中的列表项上绘制的 Drawable
android:scrollingCache		如果设为 true，该组件在滚动时将会使用绘制缓存
android:smoothScrollbar	setSmoothScrollbarEnabled(boolean)	如果设置为 false，则不在 header View 之后绘制分隔条
android:stackFromBottom		设置是否从底端开始排列列表项
android:textFilterEnabled		设置是否对列表项进行过滤。当该 AbsListView 对应的 Adapter 实现了 Filter 接口时该属性才会起作用
android:transcriptMode		设置该组件的滚动模式。该属性支持如下属性值。 ➢ disabled：关闭滚动。这是默认值 ➢ normal：当该 AbsListView 受到数据改变通知，且最后一个列表项可见时，该 AbsListView 将会滚动到底端 ➢ alwaysScroll：该 AbsListView 总会自动滚动到到底端

ListView 提供了如表 2.19 所示的常用 XML 属性。

表 2.19　ListView 的常用 XML 属性

XML 属性	说　明
android:divider	设置 List 列表项的分隔条（既可用颜色分隔，也可用 Drawable 分隔）
android:dividerHeight	设置分隔条的高度
android:entries	指定一个数组资源，Android 将根据该数组资源来生成 ListView
android:footerDividersEnabled	如果设置为 false，则不在 footer View 之前绘制分隔条
android:headerDividersEnabled	如果设置为 false，则不在 header View 之后绘制分隔条

下面通过几个实例来示范 ListView 的功能和用法。

实例：改变分隔条、基于数组的 ListView

下面的界面布局中定义了两个 ListView。

程序清单：codes\02\2.5\SimpleListViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 直接使用数组资源给出列表项 -->
<!-- 设置使用红色的分隔条 -->
<ListView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:entries="@array/books"
      android:divider="#f00"
      android:dividerHeight="2px"
      android:headerDividersEnabled="false"
      />
</LinearLayout>
```

上面的界面布局中定义了一个 ListView，并通过 android:entries 指定了列表项数组，该 ListView 还通过 android:divider 改变了列表项之间的分隔条。

上面第一个 ListView 指定了 android:entries="@array/books"，该属性值用到了数组资源，因此还需要在应用中定义一个名为 books 的数组，定义数组的资源文件如下。

程序清单：codes\02\2.5\SimpleListViewTest\res\values\arrays.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
      <string-array name="books">
            <item>疯狂 Java 讲义</item>
            <item>疯狂 Ajax 讲义</item>
            <item>疯狂 XML 讲义</item>
            <item>疯狂 Android 讲义</item>
      </string-array>
</resources>
```

使用 Activity 显示上面的 ListView，将可以看到如图 2.35 所示效果。

使用数组创建 ListView 十分简单，但这种 ListView 能定制的内容很少，甚至连每个列表项的字号大小、颜色都不能改变。

如果想对 ListView 的外观、行为进行定制，就需要把 ListView 作为 AdapterView 使用，通过 Adapter 控制每个列表项的外观和行为。



图 2.35 使用数组创建 ListView

2.5.2 Adapter 接口及实现类

Adapter 本身只是一个接口，它派生了 ListAdapter 和 SpinnerAdapter 两个子接口，其中 ListAdapter 为 AbsListView 提供列表项，而 SpinnerAdapter 为 AbsSpinner 提供列表项。Adapter

接口及其实现类的继承关系类图如图 2.36 所示。

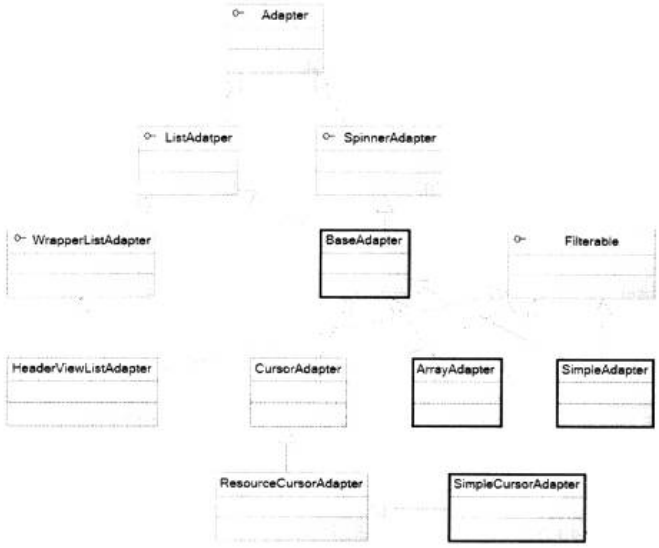


图 2.36　Adapter 及其实现类

图 2.36 所示继承类图中粗线框标出的 Adapter 是比较常用的 Adapter。从图 2.36 中所示的继承关系图可以看出，几乎所有 Adapter 都继承了 BaseAdapter，而 BaseAdapter 同时实现了 ListAdapter、SpinnerAdapter 两个接口，因此 BaseAdapter 及其子类可以同时为 AbsListView、AbsSpinner 提供列表项。

Adapter 常用的实现类如下。

- **ArrayAdapter**：简单、易用的 Adapter，通常用于将数组或 List 集合的多个值包装成多个列表项。
- **SimpleAdapter**：并不简单、功能强大的 Adapter，可用于将 List 集合的多个对象包装成多个列表项。
- **SimpleCursorAdapter**：与 SimpleAdapter 基本相似，只是用于包装 Cursor 提供的数据。
- **BaseAdapter**：通常用于被扩展。扩展 BaseAdapter 可以对各列表项进行最大限度的定制。

下面先通过 ArrayAdapter 来实现 ListView。

实例：使用 ArrayAdapter 创建 ListView

下面的实例在界面布局中定义了两个 ListView。

程序清单：codes\02\2.5\ArrayAdapterTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<!-- 设置使用红色的分隔条 -->
<ListView
    android:id="@+id/list1"
      android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:divider="#f00"
        android:dividerHeight="2px"
        android:headerDividersEnabled="false"
        />
    <!-- 设置使用绿色的分隔条 -->
    <ListView
        android:id="@+id/list2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:divider="#0f0"
        android:dividerHeight="2px"
        android:headerDividersEnabled="false"
        />
    </LinearLayout>
```

上面的布局文件定义了两个 ListView，但这两个 ListView 都没有指定 android:entries 属性，这意味着它们都需要通过 Adapter 来提供列表项。

接下来 Activity 为两个 ListView 提供 Adapter， Adapter 决定 ListView 所显示的列表项。程序如下。

程序清单：codes\02\2.5\ArrayAdapterTest\src\org\crazyit\ui\ArrayAdapterTest.java

```
public class ArrayAdapterTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ListView list1 = (ListView) findViewById(R.id.list1);
        // 定义一个数组
        String[] arr1 = { "孙悟空", "猪八戒", "牛魔王" };
        // 将数组包装为ArrayAdapter
        ArrayAdapter<String> adapter1 = new ArrayAdapter<String>
            (this, R.layout.array_item, arr1);
        // 为ListView设置Adapter
        list1.setAdapter(adapter1);
        ListView list2 = (ListView) findViewById(R.id.list2);
        // 定义一个数组
        String[] arr2 = { "Java", "Hibernate", "Spring" , "Android" };
        // 将数组包装为ArrayAdapter
        ArrayAdapter<String> adapter2 = new ArrayAdapter<String>
            (this, R.layout.checked_item, arr2);
        // 为ListView设置Adapter
        list2.setAdapter(adapter2);
    }
}
```

上面的程序中两行粗体字代码创建了两个 ArrayAdapter，创建 ArrayAdapter 时必须指定如下三个参数。

- Context：这个参数无须多说，它代表了访问整个 Android 应用的接口。几乎创建所有组件都需要传入 Context 对象。
- textViewResourceId：一个资源 ID，该资源 ID 代表一个 TextView，该 TextView 组件将作为 ArrayAdapter 的列表项组件。
- 数组或 List：该数组或 List 将负责为多个列表项提供数据。

从上面的介绍不难看出，创建 ArrayAdapter 时传入的第二个参数控制每个列表项的组件，

第三个参数则负责为列表项提供数据。该数组或 List 包含多少个元素，将生成多少个列表项，每个列表项都是 TextView 组件，TextView 组件显示的文本由数组或 List 的元素提供。

以上面的代码中第一个 ArrayAdapter 为例，该 ArrayAdapter 对应的数组为{ "孙悟空", "猪八戒", "牛魔王" }，该数组将会负责生成一个包含三个列表项的 ArrayAdapter，每个列表项的组件外观由 R.layout.array_item 布局文件（该布局文件只是一个 TextView 组件）控制，第一个 TextView 列表项显示的文本为“孙悟空”，第二个 TextView 列表项显示的文本为“猪八戒”……

上面的程序中 R.layout.array_item 布局文件如下。

程序清单：codes\02\2.5\ArrayAdapterTest\res\layout\array_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/TextView"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="24dp"
      android:padding="10px"
      android:shadowColor="#f0f"
      android:shadowDx="4"
      android:shadowDy="4"
      android:shadowRadius="2"/>
```



图 2.37　使用 ArrayAdapter 创建 ListView

上面的程序中 R.layout.checked_item 布局文件与上面的布局文件类似，具体可以参考光盘代码。

运行上面的程序将可以看到如图 2.37 所示效果。

实例：基于 ListActivity 实现列表

如果程序的窗口仅仅需要显示一个列表，则可以直接让 Activity 继承 ListActivity 来实现，ListActivity 的子类无须调用 setContentView()方法来显示某个界面，而是可以直接传入一个内容 Adapter，ListActivity 的子类就呈现出一个列表。

例如如下程序。

程序清单：codes\02\2.5\ListActivityTest\src\org\crazyit\ui\ListActivityTest.java

```
public class ListActivityTest extends ListActivity
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            // 无须使用布局文件
            String[] arr = { "孙悟空", "猪八戒", "唐僧" };
            // 创建 ArrayAdapter 对象
            ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                  android.R.layout.simple_list_item_multiple_choice, arr);
            // 设置该窗口显示列表
            setListAdapter(adapter);
      }
}
```

上面程序的 Activity 继承了 ListActivity，ListActivity 无须界面布局文件——相当于它的

布局文件中只有一个 ListView，因此只要为 ListActivity 设置 Adapter 即可。上面的程序使用 Android 提供的 R.layout.simple_list_item_multiple_choice 布局文件作为列表项组件。

运行上面的程序将看到如图 2.38 所示界面。



图 2.38 基于 ListActivity 的列表

ListActivity 的默认布局是由一个位于屏幕中心的列表组成的。实际上，开发者完全可以在 onCreate()方法中通过 setContentView(int layoutId)方法设置用户的自定义布局。

需要指出的是，在开发者指定的界面布局文件中应该包含一个 id 为"@+id/android:list"（如果是使用代码的形式，则是 android.R.id.list）的 ListView。例如包含如下代码片段：

```
<ListView android:id="@+id/android:list"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:background="#0000ff"
      android:layout_weight="1"
      android:drawSelectorOnTop="false"/>
```

实例：使用 SimpleAdapter 创建 ListView

通过 ArrayAdapter 实现 Adapter 虽然简单、易用，但 ArrayAdapter 的功能比较有限。它的每个列表项只能是 TextView。如果开发者需要实现更复杂的列表项，则可以考虑使用 SimpleAdapter。

不要被 SimpleAdapter 的名字欺骗，SimpleAdapter 并不简单，而且它的功能非常强大。ListView 的大部分应用场景，都可以通过 SimpleAdapter 来提供列表项。

例如下面先定义如下界面布局文件。

程序清单：codes\02\2.5\SimpleAdapterTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      >
<!-- 定义一个 List -->
<ListView android:id="@+id/mylist"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      />
</LinearLayout>
```

上面的界面布局中仅定义了一个 ListView，该 ListView 将会显示由 SimpleAdatper 提供的列表项。

下面是 Activity 代码。

程序清单：codes\02\2.5\SimpleAdapterTest\src\org\crazyit\ui\SimpleAdapterTest.java

```
public class SimpleAdapterTest extends Activity
{
      private String[] names = new String[]
            { "虎头", "弄玉", "李清照", "李白"};
      private String[] descs = new String[]
            { "可爱的小孩", "一个擅长音乐的女孩"
                  , "一个擅长文学的女性", "浪漫主义诗人"};
```

```
    private int[] imageIds = new int[]
        { R.drawable.tiger , R.drawable.nongyu
        , R.drawable.qingzhao , R.drawable.libai};
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建一个 List 集合，List 集合的元素是 Map
        List<Map<String, Object>> listItems =
                new ArrayList<Map<String, Object>>();
        for (int i = 0; i < names.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
            listItem.put("header", imageIds[i]);
            listItem.put("personName", names[i]);
            listItem.put("desc", descs[i]);
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this, listItems,
            R.layout.simple_item,
            new String[] { "personName", "header" , "desc"},
            new int[] { R.id.name, R.id.header , R.id.desc });
        ListView list = (ListView) findViewById(R.id.mylist);
        // 为 ListView 设置 Adapter
        list.setAdapter(simpleAdapter);
    }
}
```

上面程序的关键在于粗体字代码，粗体字代码创建了一个 SimpleAdapter。使用 SimpleAdapter 的最大难点在于创建 SimpleAdapter 对象，它需要 5 个参数，其中后面 4 个参数十分关键。

- 第 2 个参数：该参数应该是一个 List<? extends Map<String, ?>>类型的集合对象，该集合中每个 Map<String, ?>对象生成一个列表项。
- 第 3 个参数：该参数指定一个界面布局的 ID。例如此处指定了 R.layout.simple_item，这意味着使用/res/layout/simple_item.xml 文件作为列表项组件。
- 第 4 个参数：该参数应该是一个 String[]类型的参数，该参数决定提取 Map<String, ?>对象中哪些 key 对应的 value 来生成列表项。
- 第 5 个参数：该参数应该是一个 int[]类型的参数，该参数决定填充哪些组件。

从上面的程序看，listItems 是一个长度为 4 的集合，这意味着它生成的 ListView 将会包含 4 个列表项，每个列表项都是 R.layout.simple_item 对应的组件（也就是一个 LinearLayout 组件）。LinearLayout 中包含了 3 个组件：ID 为 R.id.header 的 ImageView 组件、ID 为 R.id.name、R.id.desc 的 TextView 组件，这些组件的内容由 listItems 集合提供。

R.layout.simple_item 对应的布局文件代码如下。

程序清单：codes\02\2.5\SimpleAdapterTest\res\layout\simple_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
<!-- 定义一个 ImageView，用于作为列表项的一部分 -->
<ImageView android:id="@+id/header"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp"
        />
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
    <!-- 定义一个 TextView，用于作为列表项的一部分 -->
    <TextView android:id="@+id/name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20dp"
        android:textColor="#f0f"
        android:paddingLeft="10dp"
        />
    <!-- 定义一个 TextView，用于作为列表项的一部分 -->
    <TextView android:id="@+id/desc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="14dp"
        android:paddingLeft="10dp"
        />
    </LinearLayout>
    </LinearLayout>
```

举例来说，上面 SimpleAdapter 将会生成 4 个列表项，其中第一个列表项组件是一个 LinearLayout 组件，第一个列表项的数据是{"personName"="虎头", "header"=R.id.tiger, "desc"="可爱的小孩"}Map 集合。创建 SimpleAdapter 时第 5 个参数、第 4 个参数指定使用 ID 为 R.id.name 组件显示 personName 对应的值、使用 ID 为 R.id.header 组件显示 header 对应的值、使用 ID 为 R.id.desc 显示 desc 对应的值，这样第一个列表项组件所包含的三个组件都有了显示内容。后面的每个列表项依此类推。

运行上面的程序将看到如图 2.39 所示界面。



图 2.39 使用 SimpleAdapter 生成 ListView

SimpleAdapter 同样可作为 ListActivity 的内容 Adapter，这样可以让用户方便地定制 ListActivity 所显示的列表项。

如果需要监听用户单击、选中某个列表项的事件，可以通过 AdapterView 的 setOnItemClickListener()方法为单击事件添加监听器，或通过 setOnItemSelectedListener()方法为列表项的选中事件添加监听器。

例如可以在上面的 Activity 中为 ListView 通过如下代码绑定事件监听器。

程序清单：codes\02\2.5\SimpleAdapterTest\src\org\crazyit\ui\SimpleAdapterTest.java

```
// 为 ListView 的列表项的单击事件绑定事件监听器
list.setOnItemClickListener(new OnItemClickListener()
{
    // 第 position 项被单击时激发该方法
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
            int position, long id)
    {
        System.out.println(names[position]
            + "被单击了");
    }
});
```

```
// 为 ListView 的列表项的选中事件绑定事件监听器
list.setOnItemSelectedListener(new OnItemSelectedListener()
{
    // 第 position 项被选中时激发该方法
    @Override
    public void onItemSelected(AdapterView<?> parent, View view,
            int position, long id)
    {
        System.out.println(names[position]
                + "被选中了");
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent)
    {
    }
});
```

再次运行上面程序，如果单击列表项或选中列表项，将可以看到 LogCat 控制台有如图 2.40 所示的输出。



图 2.40　ListView 的事件监听



提示：

上面的程序为 ListView 的列表项单击事件、列表项选中事件绑定了事件监听器，但事件监听器只是简单地在 LogCat 控制台输出一行内容。实际项目中我们可以在事件处理方法中做“任何”事情。不仅如此，上面绑定事件监听器的 setOnItemClickListener、setOnItemSelectedListener 方法都来自于 AdapterView，因此这种事件处理机制完全适用于 AdapterView 的其他子类。

实例：扩展 BaseAdapter 实现不存储列表项的 ListView

下面的实例将会通过扩展 BaseAdapter 来实现 Adapter，扩展 BaseAdapter 可以取得对 Adapter 最大的控制权：程序要创建多少个列表项，每个列表项的组件都由开发者来决定。

下面实例的布局文件非常简单，布局文件中只包含一个简单 ListView，布局文件代码如下。

程序清单：codes\02\2.5\BaseAdapterTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
<ListView
   android:id="@+id/myList"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>
```

该实例的 Activity 将会扩展 BaseAdapter 来实现 Adapter 对象，Activity 代码如下。

程序清单：codes\02\2.5\BaseAdapterTest\src\org\crazyit\ui\BaseAdapterTest.java

```
public class BaseAdapterTest extends Activity
{
    ListView myList;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myList = (ListView) findViewById(R.id.myList);
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                // 指定一共包含 40 个选项
                return 40;
            }
            @Override
            public Object getItem(int position)
            {
                return null;
            }
            // 重写该方法，该方法的返回值将作为列表项的 ID
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            // 重写该方法，该方法返回的 View 将作为列表框
            @Override
            public View getView(int position
                    , View convertView , ViewGroup parent)
            {
                // 创建一个 LinearLayout，并向其中添加两个组件
                LinearLayout line = new LinearLayout(BaseAdapterTest.this);
                line.setOrientation(0);
                ImageView image = new ImageView(BaseAdapterTest.this);
                image.setImageResource(R.drawable.ic_launcher);
                TextView text = new TextView(BaseAdapterTest.this);
                text.setText("第" + (position +1 ) + "个列表项");
                text.setTextSize(20);
                text.setTextColor(Color.RED);
                line.addView(image);
                line.addView(text);
                // 返回 LinearLayout 实例
                return line;
            }
        };
        myList.setAdapter(adapter);
    }
}
```

上面程序中的关键在于粗体字代码部分，粗体字代码创建了一个 BaseAdapter 对象，扩展该对象需要重写如下 4 个方法。

- getCount()：该方法的返回值控制该 Adapter 将会包含多少个列表项。
- getItem(int position)：该方法的返回值决定第 position 处的列表项的内容。

- getItemId(int position)：该方法的返回值决定第 position 处的列表项的 ID。
- getView(int position, View convertView , ViewGroup parent)：该方法的返回值决定第 position 处的列表项组件。

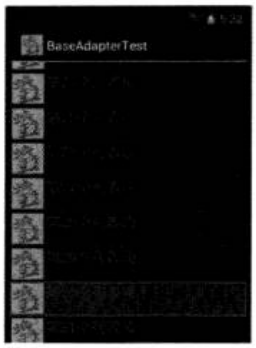


图 2.41　不存储列表项的 ListView

上面 4 个方法中最重要的是第 1 个与第 4 个。

运行上面的程序，可以看到如图 2.41 所示效果。

到目前为止，我们已经通过 ListView 介绍了 4 种实现 Adapter 的方式。表面上看，此处只是在介绍 ListView，但实际上这里介绍的知识完全适用于 AdapterView 的其他子类：GridView、Spinner、Gallery、AdapterViewFlipper 等。因此后面介绍这些组件的步骤依然是如下两步：

① 采用 4 种方式之一创建 Adapter。

② 调用 AdapterView 的 setAdapter(Adapter)方法设置 Adapter 即可。

2.5.3　自动完成文本框（AutoCompleteTextView）的功能和用法

自动完成文本框（AutoCompleteTextView）从 EditText 派生而出，实际上它也是一个文本编辑框，但它比普通编辑框多了一个功能：当用户输入一定字符之后，自动完成文本框会显示一个下拉菜单，供用户从中选择，当用户选择某个菜单项之后，AutoCompleteTextView 按用户选择自动填写该文本框。

AutoCompleteTextView 除了可使用 EditText 提供的 XML 属性和方法之外，还支持如表 2.20 所示的常用 XML 属性及相关方法。

表 2.20　AutoCompleteTextView 支持的常用 XML 属性及相关方法

XML 属性	相关方法	说　明
android:completionHint	setCompletionHint(CharSequence)	设置下拉菜单中的提示标题
android:completionHintView		设置下拉菜单中提示标题的视图
android:completionThreshold	setThreshold(int)	设置用户至少输入几个字符才会显示提示
android:dropDownAnchor	setDropDownAnchor(int)	设置下拉菜单的定位“锚点”组件，如果没有指定该属性，将使用该 TextView 本身作为定位“锚点”组件
android:dropDownHeight	setDropDownHeight(int)	设置下拉菜单的高度
android:dropDownHorizontalOffset		设置下拉菜单与文本框之间的水平偏移。下拉菜单默认与文本框左对齐
android:dropDownVerticalOffset		设置下拉菜单与文本框之间的垂直偏移。下拉菜单默认紧跟文本框
android:dropDownWidth	setDropDownWidth(int)	设置下拉菜单的宽度
android:popupBackground	setDropDownBackgroundResource(int)	设置下拉菜单的背景

使用 AutoCompleteTextView 很简单，只要为它设置一个 Adapter，该 Adapter 封装了 AutoCompleteTextView 预设的提示文本。

AutoCompleteTextView 还派生了一个子类：MultiAutoCompleteTextView，该子类的功能与 AutoCompleteTextView 基本相似，只是 MultiAutoCompleteTextView 允许输入多个提示项，多

个提示项以分隔符分隔。MultiAutoCompleteTextView 提供了 setTokenizer()方法来设置分隔符。

下面的界面布局文件中包含了 AutoCompleteTextView 和 MultiAutoCompleteTextView，布局文件代码如下。

程序清单：codes\02\2.5\AutoCompleteTextViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 定义一个自动完成文本框，
      指定输入一个字符后进行提示 -->
<AutoCompleteTextView
      android:id="@+id/auto"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:completionHint="请选择您喜欢的图书："
      android:dropDownHorizontalOffset="10dp"
      android:completionThreshold="1"/>
<!-- 定义一个 MultiAutoCompleteTextView 组件 -->
<MultiAutoCompleteTextView
    android:id="@+id/mauto"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:completionThreshold="1"/>
</LinearLayout>
```

上面的界面布局中定义了 AutoCompleteTextView 和 MultiAutoCompleteTextView，接下来在程序中为它们绑定同一个 Adapter，这意味着两个自动完成的文本框的提示项完全相同，只是它们的表现行为略有差异。

Adapter 负责为它提供提示文本。主程序如下。

程序清单：codes\02\2.5\AutoCompleteTextViewTest\src\org\crazyit\ui\AutoCompleteText- ViewTest.java

```
public class AutoCompleteTextViewTest extends Activity
{
      AutoCompleteTextView actv;
      MultiAutoCompleteTextView mauto;
      // 定义字符串数组，作为提示的文本
      String[] books = new String[]{
            "疯狂 Java 讲义",
            "疯狂 Ajax 讲义",
            "疯狂 XML 讲义",
            "疯狂 Workflow 讲义"
      };
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 创建一个 ArrayAdapter，封装数组
            ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
                  android.R.layout.simple_dropdown_item_1line, books);
            actv = (AutoCompleteTextView)findViewById(R.id.auto);
            // 设置 Adapter
            actv.setAdapter(aa);
            mauto = (MultiAutoCompleteTextView)findViewById(R.id.mauto);
            // 设置 Adapter
```

```
        mauto.setAdapter(aa);
        // 为MultiAutoCompleteTextView设置分隔符
        mauto.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
    }
}
```



图 2.42　自动完成的文本框

上面的程序中粗体字代码负责为 AutoCompleteTextView、MultiAutoCompleteTextView 设置同一个 Adapter，并为 MultiAutoCompleteTextView 设置了分隔符。

运行上面的程序将看到如图 2.42 所示界面。

2.5.4　网格视图（GridView）功能和用法

GridView 用于在界面上按行、列分布的方式来显示多个组件。

GridView 和 ListView 有共同的父类：AbsListView，因此 GridView 和 ListView 具有很高的相似性，它们都是列表项。GridView 与 ListView 的唯一区别在于：ListView 只显示一列；而 GridView 可以显示多列。从这个角度来看，ListView 相当于一种特殊的 GridView，如果让 GridView 只显示一列，那么该 GridView 就变成了 ListView。

与 ListView 类似的是，GridView 也需要通过 Adapter 来提供显示的数据：开发者可以采用上面介绍的4种方式中的任意一种来创建Adapter。不管使用哪种方式，GridView 与 ListView 的用法是基本一致的。

GridView 提供了如表 2.21 所示的常用 XML 属性。

表 2.21　GridView 常用的 XML 属性

XML 属性	相关方法	说　明
android:columnWidth	setColumnWidth(int)	设置列的宽度
android:gravity	setGravity(int)	设置对齐方式
android:horizontalSpacing	setHorizontalSpacing(int)	设置各元素之间的水平间距
android:numColumns	setNumColumns(int)	设置列数
android:stretchMode	setStretchMode(int)	设置拉伸模式
android:verticalSpacing	setVerticalSpacing(int)	设置各元素之间的垂直间距



提示：使用 GridView 时一般都应该指定 numColumns 大于 1，否则该属性的默认值为 1。如果将属性设为 1，则意味着该 GridView 只有一列，那么 GridView 就变成了 ListView。

表 2.21 所示的 android:stretchMode 属性支持如下几个属性值。

- NO_STRETCH：不拉伸。
- STRETCH_SPACING：仅拉伸元素之间的间距。
- STRETCH_SPACING_UNIFORM：表格元素本身、元素之间的间距一起拉伸。
- STRETCH_COLUMN_WIDTH：仅拉伸元素表格元素本身。

下面通过一个实例来介绍 GridView 的用法，本实例采用 SimpleAdapter 为 GridView 提供数据。

实例：带预览的图片浏览器

本实例将会采用一个 GridView 以行、列的形式来组织所有图片的预览视图。然后程序用一个 ImageView 来显示图片。

下面是本实例所使用的界面布局文件。

程序清单：codes\02\2.5\GridViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<!-- 定义一个 GridView 组件 -->
<GridView
      android:id="@+id/grid01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:horizontalSpacing="1pt"
      android:verticalSpacing="1pt"
      android:numColumns="4"
      android:gravity="center"
      />
<!-- 定义一个 ImageView 组件 -->
<ImageView android:id="@+id/imageView"
      android:layout_width="240dp"
      android:layout_height="240dp"
      android:layout_gravity="center_horizontal"
      />
</LinearLayout>
```

上面的界面布局文件中只是简单地定义了一个 GridView、一个 ImageView。定义 GridView 时指定了 android:numColumns="4"，这意味着该网格包含 4 列。那么该 GridView 包含多少行呢？这是动态改变的——正如 ListView 到底包含多少行是由该 ListView 对应的 Adapter 所决定的，GridView 到底包含多少行也是由 Adapter 决定的。

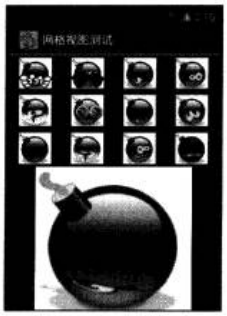
下面是主程序代码。

程序清单：codes\02\2.5\GridViewTest\src\org\crazyit\ui\GridViewTest.java

```
public class GridViewTest extends Activity
{
      GridView grid;
      ImageView imageView;
      int[] imageIds = new int[]
      {
            R.drawable.bomb5 , R.drawable.bomb6 , R.drawable.bomb7
            , R.drawable.bomb8 , R.drawable.bomb9 , R.drawable.bomb10
            , R.drawable.bomb11 , R.drawable.bomb12 , R.drawable.bomb13
            , R.drawable.bomb14 , R.drawable.bomb15 , R.drawable.bomb16
      };
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 创建一个 List 对象，List 对象的元素是 Map
            List<Map<String, Object>> listItems =
                        new ArrayList<Map<String, Object>>();
```

```
        for (int i = 0; i < imageIds.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
            listItem.put("image", imageIds[i]);
            listItems.add(listItem);
        }
        // 获取显示图片的 ImageView
        imageView = (ImageView) findViewById(R.id.imageView);
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this,
                listItems
                // 使用/layout/cell.xml 文件作为界面布局
                , R.layout.cell, new String[] { "image" },
                new int[] { R.id.image1 });
        grid = (GridView) findViewById(R.id.grid01);
        // 为 GridView 设置 Adapter
        grid.setAdapter(simpleAdapter);
        // 添加列表项被选中的监听器
        grid.setOnItemSelectedListener(new OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
                    int position, long id)
            {
                // 显示当前被选中的图片
                imageView.setImageResource(imageIds[position]);
            }
            @Override
            public void onNothingSelected(AdapterView<?> parent)
            {
            }
        });
        // 添加列表项被单击的监听器
        grid.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id)
            {
                // 显示被单击的图片
                imageView.setImageResource(imageIds[position]);
            }
        });
    }
}
```

上面的程序同样使用了 SimpleAdapter 对象作为 GridView 的 Adapter，这个 SimpleAdapter 底层保证了一个长度为 16 的 List 集合——这意味着该 GridView 一共需要显示 16 个组件，GridView 总共有 4 列，因此该 GridView 一共包含 4 行。


图 2.43　GridView 效果

上面的粗体字代码创建 SimpleAdapter 时指定使用 R.layout.cell 作为界面布局，因此还需要在/res/layout 目录下定义一个 cell.xml 界面布局文件。该文件中只包含一个简单的 ImageView 组件，此处不再给出该界面布局的代码。

运行上面的程序将可以看到程序界面上方显示了 16 个图片预览（由 GridView 提供支持），单击任何一张图片预览，下面的 ImageView 将会显示对应的图片，如图 2.43 所示。

2.5.5 可展开的列表组件（ExpandableListView）

ExpandableListView 是 ListView 的子类，它在普通 ListView 的基础上进行了扩展，它把应用中的列表项分为几组，每组里又可包含多个列表项。

ExpandableListView 的用法与普通 ListView 的用法非常相似，只是 ExpandableListView 所显示的列表项应该由 ExpandableListAdapter 提供。ExpandableListAdapter 也是一个接口，该接口下提供了如图 2.44 所示的继承关系图。

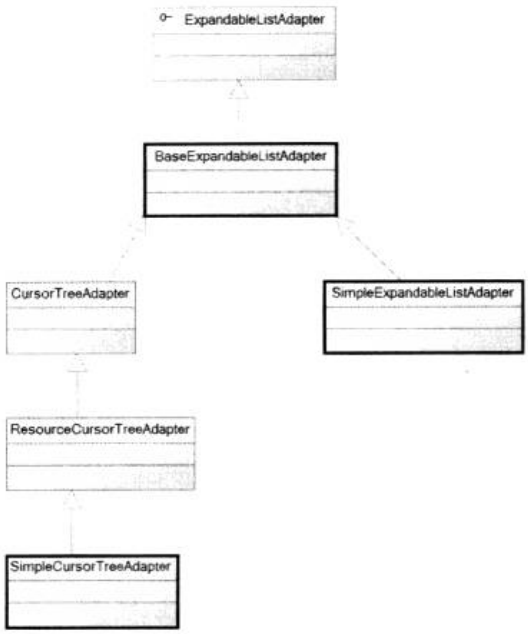


图 2.44 ExpandableListAdapter 及其子类的类图

与 Adapter 类似的是，实现 ExpandableListAdapter 也有如下三种常用方式。

- 扩展 BaseExpandableListAdapter 实现 ExpandableListAdapter
- 使用 SimpleExpandableListAdapter 将两个 List 集合包装成 ExpandableListAdapter。
- 使用 SimpleCursorTreeAdapter 将 Cursor 中的数据包装成 SimpleCursorTreeAdapter。

表 2.22 显示了 ExpandableListView 额外支持的常用 XML 属性。

表 2.22 ExpandableListView 所额外支持的常用 XML 属性

XML 属性	说 明
android:childDivider	指定各组内各子列表项之间的分隔条
android:childIndicator	显示在子列表项旁边的 Drawable 对象
android:groupIndicator	显示在组列表项旁边的 Drawable 对象

下面的程序示范了如何通过自定义 ExpandableListAdapter 为 ExpandableListView 提供列表项。该程序的界面布局文件非常简单，只是在 LinearLayout 内定义了一个 ExpandableListView，因此此处不再给出。

程序清单：codes\02\2.5\ExpandableListViewTest\src\org\crazyit\ui\ExpandableListViewTest. java

```
public class ExpandableListViewTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //创建一个 BaseExpandableListAdapter 对象
        ExpandableListAdapter adapter = new BaseExpandableListAdapter()
        {
            int[] logos = new int[]
            {
                R.drawable.p,
                R.drawable.z,
                R.drawable.t
            };
            private String[] armTypes = new String[]
                { "神族兵种", "虫族兵种", "人族兵种"};
            private String[][] arms = new String[][]
            {
                { "狂战士", "龙骑士", "黑暗圣堂", "电兵" },
                { "小狗", "刺蛇", "飞龙", "自爆飞机" },
                { "机枪兵", "护士 MM" , "幽灵" }
            };
            // 获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition, int childPosition)
            {
                return arms[groupPosition][childPosition];
            }
            @Override
            public long getChildId(int groupPosition, int childPosition)
            {
                return childPosition;
            }
            @Override
            public int getChildrenCount(int groupPosition)
            {
                return arms[groupPosition].length;
            }
            private TextView getTextView()
            {
                AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                        ViewGroup.LayoutParams.MATCH_PARENT, 64);
                TextView textView = new TextView(ExpandableListViewTest.this);
                textView.setLayoutParams(lp);
                textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
                textView.setPadding(36, 0, 0, 0);
                textView.setTextSize(20);
                return textView;
            }
            // 该方法决定每个子选项的外观
            @Override
            public View getChildView(int groupPosition, int childPosition,
                    boolean isLastChild, View convertView, ViewGroup parent)
            {
                TextView textView = getTextView();
                textView.setText(getChild(groupPosition, childPosition)
                        .toString());
```

```
            return textView;
        }
        // 获取指定组位置处的组数据
        @Override
        public Object getGroup(int groupPosition)
        {
            return armTypes[groupPosition];
        }
        @Override
        public int getGroupCount()
        {
            return armTypes.length;
        }
        @Override
        public long getGroupId(int groupPosition)
        {
            return groupPosition;
        }
        // 该方法决定每个组选项的外观
        @Override
        public View getGroupView(int groupPosition, boolean isExpanded,
                View convertView, ViewGroup parent)
        {
            LinearLayout ll = new LinearLayout(ExpandableListViewTest. this);
            ll.setOrientation(0);
            ImageView logo = new ImageView(ExpandableListViewTest.this);
            logo.setImageResource(logos[groupPosition]);
            ll.addView(logo);
            TextView textView = getTextView();
            textView.setText(getGroup(groupPosition).toString());
            ll.addView(textView);
            return ll;
        }
        @Override
        public boolean isChildSelectable(int groupPosition,
                int childPosition)
        {
            return true;
        }
        @Override
        public boolean hasStableIds()
        {
            return true;
        }
    };
    ExpandableListView expandListView = (ExpandableListView) findViewById
    (R.id.list);
    expandListView.setAdapter(adapter);
  }
}
```

上面程序的关键代码就是扩展 BaseExpandableListAdapter 来实现 ExpandableListAdapter，当扩展 BaseExpandableListAdapter 时，关键是实现如下 4 个方法。

- getGroupCount()：该方法返回包含的组列表项的数量。
- getGroupView()：该方法返回的 View 对象将作为组列表项。
- getChildrenCount()：该方法返回特定组所包含的子列表项的数量。
- getChildView()：该方法返回的 View 对象将作为特定组、特定位置的子列表项。

上面的程序中 getChildView()方法返回了一个普通 TextView，因为每个子列表项都是一个普通文本框。而 getGroupView 方法则返回了一个 LinearLayout 对象，该 LinearLayout 对象里包含一个 ImageView 和一个 TextView。因此每个组列表项都由图片和文本组成。

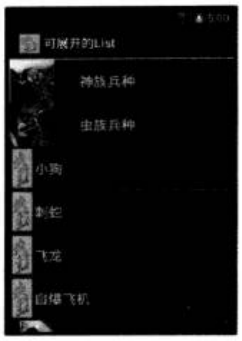


图 2.45　ExpandableListView 效果

运行上面的程序将看到如图 2.45 所示界面。

前面我们介绍过，上面 ExpandableListView 的每个子列表项都应该只是普通文本，那么为何从图 2.45 中可以看到每个子列表项左边都有一个图片呢？其实这是因为我们在定义 ExpandableListView 时指定了 android:childIndicator="@drawable/ ic_launcher"，它会自动在每个子列表项旁边添加一个图片。

2.5.6　Spinner 的功能和用法

Spinner 组件与 Swing 编程 Spinner 不同，此处的 Spinner 其实就是一个列表选择框。不过 Android 的列表选择框并不是需要显示下拉列表的，而是相当于弹出一个菜单供用户选择。

Spinner 与 Gallery 都继承了 AbsSpinner，AbsSpinner 继承了 AdapterView，因此它也表现出 AdapterView 的特征：只要为 AdapterView 提供 Adapter 即可。

Spinner 支持如表 2.23 所示的常用 XML 属性。

表 2.23　Spinner 的常用 XML 属性

XML 属性	相关方法	说　明
android:entries		使用数组资源设置该下拉列表框的列表项目
android:dropDownHorizontalOffset	setDropDownHorizontalOffset(int)	设置下拉列表框的水平偏移距
android:dropDownVerticalOffset	setDropDownVerticalOffset(int)	设置下拉列表框的垂直偏移距
android:dropDownWidth	setDropDownWidth(int)	设置下拉列表框的宽度
android:popupBackground	android:popupBackground	设置下拉列表框的背景色
android:prompt		设置该列表选择框的提示信息



备注： android:entries 属性并不是 Spinner 定义的，而是在 AbsSpinner 中定义的，因此后面介绍的 Gallery（继承了 AbsSpinner）也支持该 XML 属性。

实例：让用户选择

如果开发者使用 Spinner 时已经可以确定下拉列表框里的列表项，则完全不需要编写代码，只要为 Spinner 指定 android:entries 属性即可实现 Spinner；如果程序需要在运行时动态地决定 Spinner 的列表项，或程序需要对 Spinner 的列表项进行定制，则可使用 Adapter 为 Spinner 提供列表项。

如下布局文件中定义了两个 Spinner 组件，其中第一个 Spinner 组件指定了 android:entries 属性，第二个 Spinner 组件没有指定 android:entries 属性，因此需要在 Activity 中为它设置 Adapter。

该实例的界面布局文件如下。

程序清单：codes\02\2.5\SpinnerTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<!-- 定义了一个 Spinner 组件，
      指定显示该 Spinner 组件的数组 -->
<Spinner
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:entries="@array/books"
      android:prompt="@string/tip"
      />
<Spinner
    android:id="@+id/spinner"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:prompt="@string/tip"
      />
</LinearLayout>
```

上面的界面布局文件中第一个 Spinner 使用@array/books 指定数组资源，因此需要在 res/value 目录下使用 XML 文件来定义一份数组资源，该数组资源文件的内容如下。

程序清单：codes\02\2.5\SpinnerTest\res\values\arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <string-array name="books">
            <item>疯狂 Java 讲义</item>
            <item>疯狂 Ajax 讲义</item>
            <item>疯狂 XML 讲义</item>
      </string-array>
 </resources>
```

上面的布局文件中第二个 Spinner 没有指定 android:entries 属性，Activity 将会采用 ArrayAdapter 为该 Spinner 提供列表项。下面是 Activity 的代码。

程序清单：codes\02\2.5\SpinnerTest\src\org\crazyit\ui\SpinnerTest.java

```
public class SpinnerTest extends Activity
{
      Spinner spinner;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取界面布局文件中的 Spinner 组件
            spinner = (Spinner) findViewById(R.id.spinner);
            String[] arr = { "孙悟空", "猪八戒", "唐僧" };
            // 创建 ArrayAdapter 对象
            ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                  android.R.layout.simple_list_item_multiple_choice, arr);
            // 为 Spinner 设置 Adapter
            spinner.setAdapter(adapter);
      }
}
```

上面的粗体字代码创建了一个简单的 ArrayAdapter，并调用了 Spinner 的 setAdapter(adapter)为之设置 Adpater。运行该程序，将看到如图 2.46 所示的界面。



图 2.46　Spinner 的效果

➤➤2.5.7　画廊视图（Gallery）的功能和用法

Gallery 与 Spinner 组件有共同的父类：AbsSpinner，表明 Gallery 和 Spinner 都是一个列表框。它们之间的区别在于 Spinner 显示的是一个垂直的列表选择框，而 Gallery 显示的是一个水平的列表选择框。Gallery 与 Spinner 还有一个区别：Spinner 的作用是供用户选择，而 Gallery 则允许用户通过拖动来查看上一个、下一个列表项。

Gallery 额外提供了如表 2.24 所示的 XML 属性。

表 2.24　Gallery 常用的 XML 属性及相关方法

XML 属性	相关方法	说　明
android:animationDuration	setAnimationDuration(int)	设置图片切换时的动画持续时间
android:gravity	setGravity(int)	设置对齐方式
android:spacing	setSpacing(int)	设置图片之间的间距
android:unselectedAlpha	setUnselectedAlpha(float)	设置没有选中的图片的透明度

Gallery 本身的用法非常简单——基本上与 Spinner 的用法相似，只要为它提供一个内容 Adapter 即可，该 Adapter 的 getView 方法所返回的 View 将作为 Gallery 列表的列表项。如果程序需要监控到 Gallery 选择项的改变，通过为 Gallery 添加 OnItemSelectedListener 监听器即可实现。

下面通过一个实例来介绍 Gallery 的用法。

实例：“幻灯片”式图片浏览器

本实例也是带预览的图片浏览器，但本实例的界面更加友好，因为本例采用 Gallery 作为图片预览器，Gallery 会生成一个“水平列表”，每个列表项就是一张图片预览，而且 Gallery 生成的“水平列表”可以让用户通过拖动来切换列表项。

下面是本应用的界面布局文件。

程序清单：codes\02\2.5\GallaryTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<!-- 定义一个 ImageView 组件 -->
<ImageView android:id="@+id/imageView"
      android:layout_width="320dp"
      android:layout_height="320dp"
      />
<!-- 定义一个 Gallery 组件 -->
<Gallery android:id="@+id/gallery"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="25dp"
      android:unselectedAlpha="0.6"
```

```
        android:spacing="2pt"
        />
</LinearLayout>
```

上面的界面布局十分简单，仅仅定义了两个组件：ImageView 和 Gallery 组件。接下来的主程序也很简单：为 Gallery 传入一个 Adapter 对象。这样 Gallery 即可正常工作。

为了让 ImageView 可以显示 Gallery 中选中的图片，为 Gallery 绑定 OnItemSelectedListener 监听器即可。程序代码如下。

程序清单：codes\02\2.5\GallaryTest\src\org\crazyit\ui\GalleryTest.java

```
public class GallaryTest extends Activity
{
    int[] imageIds = new int[]
    {
        R.drawable.shuangzi, R.drawable.shuangyu,
        R.drawable.chunv, R.drawable.tiancheng, R.drawable.tianxie,
        R.drawable.sheshou, R.drawable.juxie, R.drawable.shuiping,
        R.drawable.shizi, R.drawable.baiyang, R.drawable.jinniu,
        R.drawable.mojie };
    Gallery gallery;
    ImageView imageView;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        gallery = (Gallery) findViewById(R.id.gallery);
        // 获取显示图片的 ImageView 对象
        imageView = (ImageView) findViewById(R.id.imageView);
        // 创建一个 BaseAdapter 对象，该对象负责提供 Gallery 所显示的列表项
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                return imageIds.length;
            }
            @Override
            public Object getItem(int position)
            {
                return position;
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            // 该方法返回的 View 代表了每个列表项
            @Override
            public View getView(int position, View convertView, ViewGroup parent)
            {
                // 创建一个 ImageView
                ImageView imageView = new ImageView(GallaryTest.this);
                imageView.setImageResource(imageIds[position]);
                // 设置 ImageView 的缩放类型
                imageView.setScaleType(ImageView.ScaleType.FIT_XY);
                // 为 imageView 设置布局参数
                imageView.setLayoutParams(new Gallery.LayoutParams(75, 100));
                TypedArray typedArray = obtainStyledAttributes(
```

```
                                    R.styleable.Gallery);
                            imageView.setBackgroundResource(typedArray.getResourceId(
        R.styleable.Gallery_android_galleryItemBackground, 0));
                            return imageView;
                        }
                    };
                    gallery.setAdapter(adapter);
                    gallery.setOnItemSelectedListener(new OnItemSelectedListener()
                    {
                        // 当Gallery选中项发生改变时触发该方法
                        @Override
                        public void onItemSelected(AdapterView<?> parent, View view,
                                int position, long id)
                        {
                            imageView.setImageResource(imageIds[position]);
                        }
                        @Override
                        public void onNothingSelected(AdapterView<?> parent)
                        {
                        }
                    });
                }
            }
```

上面的程序中粗体字代码创建了一个 BaseAdapter 对象，该 Adapter 将负责为 Gallery 提供列表项。运行上面的程序，将可以看到如图 2.47 所示界面。

注意：

Android 已经不再推荐使用 Gallery 组件，而是推荐使用其他水平滚动组件如 HorizontalScrollView 和 ViewPager 来代替 Gallery 组件。因此在新版本的 Android 平台上应该尽量少用 Gallery 组件。



图 2.47　Gallery 画廊

2.5.8　AdapterViewFlipper 的功能与用法

AdapterViewFilpper 继承了 AdapterViewAnimator，它也会显示 Adapter 提供的多个 View 组件，但它每次只能显示一个 View 组件，程序可通过 showPrevious()和 showNext()方法控制该组件显示上一个、下一个组件。

AdapterViewFilpper 可以在多个 View 切换过程中使用渐隐渐现的动画效果，除此之外，还可以调用该组件的 startFlipping()控制它“自动播放”下一个 View 组件。

AdapterViewAnimator 可以指定如表 2.25 所示的 XML 属性。

表 2.25　AdapterViewAnimator 支持的 XML 属性

XML 属性	说　明
android:animateFirstView	设置显示该组件的第一个 View 时是否使用动画
android:inAnimation	设置组件显示时使用的动画
android:loopViews	设置循环到最后一个组件后是否自动“转头”到第一个组件
android:outAnimation	设置组件隐藏时使用的动画

AdapterViewFilpper 可以额外指定如表 2.26 所示的 XML 属性。

表 2.26 AdapterViewFilpper 支持的 XML 属性

XML 属性	相关方法	说 明
android:autoStart	startFlipping()	设置显示该组件是否自动播放
android:flipInterval	setFlipInterval(int)	设置自动播放的时间间隔

实例：自动播放的图片库

下面的实例示范了如何使用 AdapterViewFlipper 开发自动播放的图片库，该实例的界面上除了包含一个 AdapterViewFilpper 之外，还包含三个按钮，用于控制显示“上一张”、“下一张”和“自动播放”。为了控制 AdapterViewFilpper 要显示的多个列表项，程序为 AdapterViewFilpper 设置一个 Adapter 即可。

下面是该实例的 XML 布局文件。

程序清单：codes\02\2.5\AdapterViewFlipperTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <AdapterViewFlipper
        android:id="@+id/flipper"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:flipInterval="5000"
        android:layout_alignParentTop="true"/>
    <Button
       android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:onClick="prev"
        android:text="上一个"
        />
    <Button
       android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:onClick="next"
        android:text="下一个"
        />
    <Button
       android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:onClick="auto"
        android:text="自动播放"
        />
</RelativeLayout>
```

上面的粗体字代码定义了一个 AdapterViewFlipper 组件，并为三个按钮指定了事件处理方法。该实例的 Activity 会采用扩展 BaseAdapter 的方式来实现自己的 Adapter，并为

AdapterViewFlipper 组件设置 Adapter。下面是该 Activity 的代码。

程序清单：codes\02\2.5\AdapterViewFlipperTest\src\org\crazyit\ui\AdapterViewFlipperTest.java

```
public class AdapterViewFlipperTest extends Activity
{
    int[] imageIds = new int[]
    {
        R.drawable.shuangzi, R.drawable.shuangyu,
        R.drawable.chunv, R.drawable.tiancheng, R.drawable.tianxie,
        R.drawable.sheshou, R.drawable.juxie, R.drawable.shuiping,
        R.drawable.shizi, R.drawable.baiyang, R.drawable.jinniu,
        R.drawable.mojie };
    AdapterViewFlipper flipper;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        flipper = (AdapterViewFlipper) findViewById(R.id.flipper);
        // 创建一个 BaseAdapter 对象，该对象负责提供 Gallery 所显示的列表项
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                return imageIds.length;
            }
            @Override
            public Object getItem(int position)
            {
                return position;
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            // 该方法返回的 View 代表了每个列表项
            @Override
            public View getView(int position, View convertView, ViewGroup parent)
            {
                // 创建一个 ImageView
                ImageView imageView = new ImageView(AdapterViewFlipperTest.
                this);
                imageView.setImageResource(imageIds[position]);
                // 设置 ImageView 的缩放类型
                imageView.setScaleType(ImageView.ScaleType.FIT_XY);
                // 为 imageView 设置布局参数
                imageView.setLayoutParams(new LayoutParams(
                        LayoutParams.MATCH_PARENT,  LayoutParams.MATCH_
                        PARENT));
                return imageView;
            }
        };
        flipper.setAdapter(adapter);
    }
    public void prev(View source)
    {
        // 显示上一个组件
        flipper.showPrevious();
        // 停止自动播放
```

```
            flipper.stopFlipping();
        }
        public void next(View source)
        {
            // 显示下一个组件。
            flipper.showNext();
            // 停止自动播放
            flipper.stopFlipping();
        }
        public void auto(View source)
        {
            // 开始自动播放
            flipper.startFlipping();
        }
}
```

上面的程序中粗体字代码调用了 AdapterViewFlipper 的 showPrevious()、showNext()方法来控制该组件显示上一个、下一个组件，并调用了 startFlipping()方法控制自动播放。

运行该程序，可以看到如图 2.48 所示界面。

在图 2.48 中单击“自动播放”按钮，将可以看到 AdapterViewFlipper 每隔 5 秒更换一张图片，切换图片时会使用渐隐渐显效果。



图 2.48 使用 AdapterViewFlipper 实现自动播放图片

2.5.9 StackView 的功能与用法

StackView 也是 AdapterViewAnimator 的子类，它也用于显示 Adapter 提供的系列 View。StackView 将会以“堆叠（Stack）”方式来显示多个列表项。

为了控制 StackView 显示的 View 组件，StackView 提供了如下两种控制方式。

- 拖走 StackView 中处于顶端的 View，下一个 View 将会显示出来。将上一个 View 拖进 StackView，将使之显示出来。
- 通过调用 StackView 的 showNext()、showPrevious()控制显示上一个、上一个组件。

下面的实例示范了 StackView 的功能和用法。

实例：叠在一起的图片

该实例会使用 StackView 将照片叠在一起，并允许用户通过拖动或单击按钮来显示上一张、下一张图片。该实例的布局文件如下。

程序清单：codes\02\2.5\StackViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <StackView
        android:id="@+id/mStackView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:loopViews="true" />
```

```
        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="上一个"
                android:onClick="prev"/>
            <Button
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="下一个"
                android:onClick="next"/>
        </LinearLayout>
    </LinearLayout>
```

上面的布局文件中定义了一个 StackView 组件，该 StackView 将会以“叠”的方式显示多个 View 组件。与所有 AdapterView 类似的是，只要为 StackView 设置 Adapter 即可。

下面 Activity 将会创建一个 SimpleAdapte 作为 StackView 的 Adapter，并为布局文件中的两个按钮的 onClick 事件提供处理方法。下面是该 Activity 的代码。

程序清单：codes\02\2.5\StackViewTest\src\org\crazyit\ui\StackViewTest.java

```
public class StackViewTest extends Activity
{
    StackView stackView;
    int[] imageIds = new int[]
    {
        R.drawable.bomb5 , R.drawable.bomb6 , R.drawable.bomb7
        , R.drawable.bomb8 , R.drawable.bomb9 , R.drawable.bomb10
        , R.drawable.bomb11 , R.drawable.bomb12       , R.drawable.bomb13
        , R.drawable.bomb14 , R.drawable.bomb15 , R.drawable.bomb16
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        stackView = (StackView) findViewById(R.id.mStackView);
        // 创建一个 List 对象，List 对象的元素是 Map
        List<Map<String, Object>> listItems =
                new ArrayList<Map<String, Object>>();
        for (int i = 0; i < imageIds.length; i++)
        {
            Map<String, Object> listItem = new HashMap<String, Object>();
            listItem.put("image", imageIds[i]);
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this,
                listItems
                // 使用/layout/cell.xml 文件作为界面布局
                , R.layout.cell, new String[] { "image" },
                new int[] { R.id.image1 });
        stackView.setAdapter(simpleAdapter);
    }
    public void prev(View view)
    {
        // 显示上一个组件
        stackView.showPrevious();
```

```
        }
        public void next(View view)
        {
            // 显示下一个组件
            stackView.showNext();
        }
}
```

上面的 Activity 代码中粗体字代码创建了一个 SimpleAdapter，并将这个 SimpleAdapter 设置为该 StackView 的 Adapter，这样该 StackView 将会显示该 SimpleAdapter 包含的系列 View 组件。

运行该程序，并拖动 ViewStack 顶端的 View 组件，可以看到如图 2.49 所示效果。

图 2.49 下方这张图片是正在被拖动的图片，这张图片显示的“动画”效果正是 StackView 支持的效果。

2.6 第 5 组 UI 组件：ProgressBar 及其子类

ProgressBar 组件也是一组重要的组件，ProgressBar 本身代表了进度条组件，它还派生了两个常用的组件：SeekBar 和 RatingBar。ProgressBar 及其子类在用法上十分相似，只是显示界面有一定的区别，因此本节把它们归为一类，针对它们的共性集中讲解，并突出介绍它们的区别。

ProgressBar 及其子类的继承关系图如图 2.50 所示。



图 2.49 StackView 测试

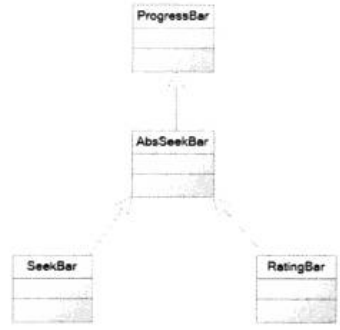


图 2.50 ProgressBar 及其子类的继承关系图

2.6.1 进度条（ProgressBar）的功能与用法

进度条也是 UI 界面中的一种非常实用的组件，通常用于向用户显示某个耗时操作完成的百分比。进度条可以动态地显示进度，因此避免长时间地执行某个耗时操作时，让用户感觉程序失去了响应，从而更好地提高用户界面的友好性。

Android 支持几种风格的进度条，通过 style 属性可以为 ProgressBar 指定风格。该属性可支持如下几个属性值。

- @android:style/Widget.ProgressBar.Horizontal：水平进度条。
- @android:style/Widget.ProgressBar.Inverse：普通大小的环形进度条。
- @android:style/Widget.ProgressBar.Large：大环形进度条。
- @android:style/Widget.ProgressBar.Large.Inverse：大环形进度条。

- @android:style/Widget.ProgressBar.Small：小环形进度条。
- @android:style/Widget.ProgressBar.Small.Inverse：小环形进度条。

除此之外，ProgressBar 还支持如表 2.27 所示的常用 XML 属性。

表 2.27　ProgressBar 常用的 XML 属性

XML 属性	说　明
android:max	设置该进度条的最大值
android:progress	设置该进度条的已完成进度值
android:progressDrawable	设置该进度条的轨道对应的 Drawable 对象
android:indeterminate	该属性设为 true，设置进度条不精确显示进度
android:indeterminateDrawable	设置绘制不显示进度的进度条的 Drawable 对象
android:indeterminateDuration	设置不精确显示进度的持续时间

表 2.27 中 android:progressDrawable 用于指定进度条的轨道的绘制形式，该属性可指定为一个 LayerDrawable 对象（该对象可通过在 XML 文件中用<layer-list>元素进行配置）的引用。

ProgressBar 提供了如下方法来操作进度。

- setProgress(int)：设置进度的完成百分比。
- incrementProgressBy(int)：设置进度条的进度增加或减少。当参数为正数时进度增加；当参数为负数时进度减少。

下面的程序简单示范了进度条的用法，该程序的界面布局文件只是定义了几个简单的进度条，并指定 style 属性为@android:style/Widget.ProgressBar.Horizontal，即水平进度条。界面布局文件如下。

程序清单：codes\02\2.6\ProgressBarTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent"      >
<LinearLayout
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="wrap_content">
<!-- 定义一个大环形进度条 -->
<ProgressBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      style="@android:style/Widget.ProgressBar.Large"/>
<!-- 定义一个中等大小的环形进度条 -->
<ProgressBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<!-- 定义一个小环形进度条 -->
<ProgressBar
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      style="@android:style/Widget.ProgressBar.Small"/>
</LinearLayout>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="任务完成的进度"/>
```

```
<!-- 定义一个水平进度条 -->
<ProgressBar
      android:id="@+id/bar"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:max="100"
      style="@android:style/Widget.ProgressBar.Horizontal"/>
<!-- 定义一个水平进度条，并改变轨道外观 -->
<ProgressBar
      android:id="@+id/bar2"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:max="100"
      android:progressDrawable="@drawable/my_bar"
      style="@android:style/Widget.ProgressBar.Horizontal"/>
</LinearLayout>
```

上面的布局文件中先定义了三个环形进度条，这种环形进度条无法显示进度，它只是显示一个不断旋转的图片。布局文件的后面定义的两个进度条的最大值为 100，第一个进度条的样式为水平进度条；第二个进度条的外观被定义为@drawable/my_bar，因此还需要在 drawable-mdpi 中定义如下文件。

程序清单：codes\02\2.6\ProgressBarTest\res\drawable-mdpi\my_bar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
      <!-- 定义轨道的背景 -->
      <item android:id="@android:id/background"
            android:drawable="@drawable/no" />
      <!-- 定义轨道上已完成部分的样式 -->
      <item android:id="@android:id/progress"
            android:drawable="@drawable/ok" />
</layer-list>
```

下面的主程序用一个填充数组的任务模拟了耗时操作，并以进度条来标识任务的完成百分比，主程序如下。

程序清单：codes\02\2.6\ProgressBarTest\src\org\crazyit\ui\ProgressBarTest.java

```
public class ProgressBarTest extends Activity
{
      // 该程序模拟填充长度为 100 的数组
      private int[] data = new int[100];
      int hasData = 0;
      // 记录 ProgressBar 的完成进度
      int status = 0;
      ProgressBar bar , bar2;
      // 创建一个负责更新的进度的 Handler
      Handler mHandler = new Handler()
      {
            @Override
            public void handleMessage(Message msg)
            {
                  // 表明消息是由该程序发送的
                  if (msg.what == 0x111)
                  {
                        bar.setProgress(status);
                        bar2.setProgress(status);
                  }
            }
```

```
};
@Override
public void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      bar = (ProgressBar) findViewById(R.id.bar);
      bar2 = (ProgressBar) findViewById(R.id.bar2);
      // 启动线程来执行任务
      new Thread()
      {
            public void run()
            {
                  while (status < 100)
                  {
                        // 获取耗时操作的完成百分比
                        status = doWork();
                        // 发送消息
                        mHandler.sendEmptyMessage(0x111);
                  }
            }
      }.start();
}
// 模拟一个耗时的操作
public int doWork()
{
      // 为数组元素赋值
      data[hasData++] = (int) (Math.random() * 100);
      try
      {
            Thread.sleep(100);
      }
      catch (InterruptedException e)
      {
            e.printStackTrace();
                  }
                  return hasData;
            }
      }
```

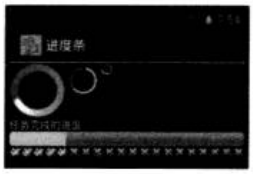


图 2.51　进度条

上面的程序中粗体字代码用于修改进度条的完成进度。运行上面的程序，将看到如图 2.51 所示界面。

实例：显示在标题上的进度条

还有一种进度条，可以直接在窗口标题上显示，这种进度条甚至不需要使用 ProgressBar 组件，它是直接由 Activity 的方法启用的。为了在窗口上显示进度条，需要经过如下两步。

① 调用 Activity 的 requestWindowFeature()方法，该方法根据传入的参数可启用特定的窗口特征，例如传入 Window.FEATURE_INDETERMINATE_PROGRESS 在窗口标题上显示不带进度的进度条；传入 Window.FEATURE_PROGRESS 则显示带进度的进度条。

② 调用 Activity 的 setProgressBarVisibility(boolean) 或 setProgressBarIndeterminate Visibility(boolean)方法即可控制进度条的显示和隐藏。

例如下面的程序，界面布局中仅仅定义了两个按钮，此处不再给出界面布局文件。该程序的 Activity 类代码如下。

程序清单：codes\02\2.6\TitleProgressBar\src\org\crazyit\ui\TitleProgressBar.java

```
public class TitleProgressBar extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        //设置窗口特征：启用显示进度的进度条
        requestWindowFeature(Window.FEATURE_PROGRESS); //①
        //设置窗口特征：启用不显示进度的进度条
        requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS); //②
        setContentView(R.layout.main);
        Button bn1 = (Button)findViewById(R.id.bn1);
        Button bn2 = (Button)findViewById(R.id.bn2);
        bn1.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                //显示不带进度的进度条
                setProgressBarIndeterminateVisibility(true);
                //显示带进度的进度条
                setProgressBarVisibility(true);
                //设置进度条的进度
                setProgress(4500);
            }
        });
        bn2.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                //隐藏不带进度的进度条
                setProgressBarIndeterminateVisibility(false);
                //隐藏带进度的进度条
                setProgressBarVisibility(false);
            }
        });
    }
}
```

上面的程序中①号代码控制窗口标题上显示带进度的进度条，而②号代码则控制窗口标题上显示不带进度的进度条。程序中两个按钮主要用于控制进度条的显示、隐藏。

如果程序启动窗口上显示带进度的进度条，则可看到如图 2.52 所示界面。

如果程序启动窗口上显示不带进度的进度条，则可看到如图 2.53 所示界面。



图 2.52 显示在标题上的带进度的进度条

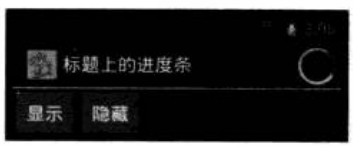


图 2.53 显示在标题上的不带进度的进度条

2.6.2 拖动条（SeekBar）的功能和用法

拖动条和进度条非常相似，只是进度条采用颜色填充来表明进度完成的程度，而拖动条则通过滑块的位置来标识数值——而且拖动条允许用户拖动滑块来改变值，因此拖动条通常用于对系统的某种数值进行调节，比如调节音量等。



提示：

由于拖动条 SeekBar 继承了 ProgressBar，因此 ProgressBar 所支持的 XML 属性和方法完全适用于 SeekBar。

SeekBar 允许用户改变拖动条的滑块外观，改变滑块外观通过如下属性来指定。

- android:thumb：指定一个 Drawable 对象，该对象将作为自定义滑块。

为了让程序能响应拖动条滑块位置的改变，程序可以考虑为它绑定一个 OnSeekBarChangeListener 监听器。

下面通过一个实例来示范 SeekBar 的功能和用法。

实例：通过拖动滑块来改变图片的透明度

该程序的界面布局中需要两个组件：一个 ImageView 用于显示图片，一个 SeekBar 用于动态改变图片的透明度。界面布局文件如下。

程序清单：codes\02\2.6\SeekBarTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<ImageView
      android:id="@+id/image"
      android:layout_width="fill_parent"
      android:layout_height="240px"
      android:src="@drawable/lijiang"
      />
<!-- 定义一个拖动条，并改变它的滑块外观 -->
<SeekBar
      android:id="@+id/seekbar"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:max="255"
      android:progress="255"
      android:thumb="@drawable/ic_launcher"
      />
</LinearLayout>
```

上面的程序中粗体字代码定义了该拖动条的最大值、当前值都是 255，并通过指定 android:thumb 属性来改变拖动条上滑块的外观。

该示例的主程序比较简单，程序只要为拖动条绑定一个监听器，当滑块位置发生改变时动态改变 ImageView 的透明度即可。主程序如下。

程序清单：codes\02\2.6\SeekBarTest\src\org\crazyit\ui\SeekBarTest.java

```
public class SeekBarTest extends Activity
{
      ImageView image;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
```

```
        image = (ImageView) findViewById(R.id.image);
        SeekBar seekBar = (SeekBar) findViewById(R.id.seekbar);
        seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener()
        {
            // 当拖动条的滑块位置发生改变时触发该方法
            @Override
            public void onProgressChanged(SeekBar arg0, int progress,
                    boolean fromUser)
            {
                // 动态改变图片的透明度
                image.setAlpha(pogress);
            }
            @Override
            public void onStartTrackingTouch(SeekBar bar)
            {
            }
            @Override
            public void onStopTrackingTouch(SeekBar bar)
            {
            }
        });
    }
}
```

上面的粗体字代码就是监听拖动条上滑块位置发生改变的关键代码：当滑块位置发生改变时，ImageView 的透明度将变为该拖动条的当前数值。运行上面的程序将看到如图 2.54 所示界面。

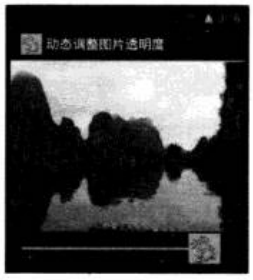


图 2.54 拖动滑块改变图片的透明度

2.6.3 星级评分条（RatingBar）的功能和用法

星级评分条与拖动条有相同的父类：AbsSeekBar，因此它们十分相似。实际上星级评分条与拖动条的用法、功能都十分接近：它们都允许用户通过拖动来改变进度。RatingBar 与 SeekBar 最大区别在于：RatingBar 通过星星来表示进度。

表 2.28 显示了星级评分条所支持的常见 XML 属性。

表 2.28 RatingBar 支持的常见 XML 属性

XML 属性	说 明
android:isIndicator	设置该星级评分条是否允许用户改变（true 为不允许修改）
android:numStars	设置该星级评分条总共有多少个星级
android:rating	设置该星级评分条默认的星级
android:stepSize	设置每次最少需要改变多少个星级

为了让程序能响应星级评分条评分的改变，程序可以考虑为它绑定一个 OnRatingBarChangeListener 监听器。

下面通过一个实例来示范 RatingBar 的功能和用法。

实例：通过星级改变图片的透明度

该程序其实只是前一个程序的简单改变，只是将上面程序中的 SeekBar 组件改为使用

RatingBar。下面是界面布局中关于 RatingBar 的代码片段。

程序清单：codes\02\2.6\RatingBarTest\res\layout\main.xml

```
<!-- 定义一个星级评分条 -->
<RatingBar
      android:id="@+id/rating"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:numStars="5"
      android:max="255"
      android:progress="255"
      android:stepSize="0.5"
      />
```

上面的界面布局中指定了该星级评分条的最大值为 255，当前进度为 255——其中两个属性都来自于 ProgressBar 组件，这没有任何问题，因为 RatingBar 本来就是一个特殊的 ProgressBar。

主程序只要为 RatingBar 绑定事件监听器即可，监听星级评分条的星级改变。下面的主程序为星级评分条绑定监听器的代码。

程序清单：codes\02\2.6\RatingBarTest\src\org\crazyit\ui\RatingBarTest.java

```
ratingBar.setOnRatingBarChangeListener(new OnRatingBarChangeListener()
{
      // 当拖动条的滑块位置发生改变时触发该方法
      @Override
      public void onRatingChanged(RatingBar arg0, float rating,
                  boolean fromUser)
      {
            // 动态改变图片的透明度，其中 255 是星级评分条的最大值
            // 5 个星星就代表最大值 255
            image.setAlpha((int) (rating * 255 / 5));
      }
});
```



图 2.55　星级评分条改变图片的透明度

由于上面定义 RatingBar 时指定了 android:stepSize="0.5"，因此该星级评分条中星级的最小变化为 0.5，也就是最少要变化半个星级。运行上面的程序将看到如图 2.55 所示界面。

2.7　第 6 组 UI 组件：ViewAnimator 及其子类

ViewAnimator 是一个基类，它继承了 FrameLayout，因此它表现出 FrameLayout 的特征，可以将多个 View 组件“叠”在一起。ViewAnimator 额外增加的功能正如它的名字所暗示的，ViewAnimator 可以在 View 切换时表现出动画效果。

ViewAnimator 及其子类的继承关系如图 2.56 所示。

ViewAnimator 及其子类也是一组非常重要的 UI 组件，这种组件的主要功能是增加动画效果，从而使界面更加“炫”。使用 ViewAnimator 时可以指定如表 2.29 所示的常见 XML 属性。

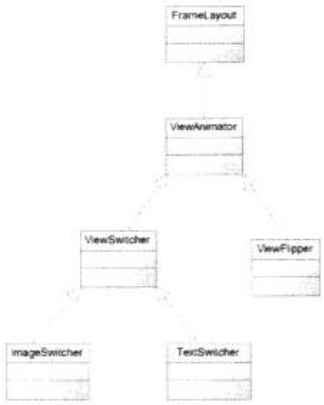


图 2.56 ViewAnimator 及其子类的继承关系

表 2.29 ViewAnimator 支持的常见 XML 属性

XML 属性	说 明
android:animateFirstView	设置 ViewAnimator 显示第一个 View 组件时是否使用动画
android:inAnimation	设置 ViewAnimator 显示组件时使用的动画
android:outAnimation	设置 ViewAnimator 隐藏组件时使用的动画

实际项目中往往会使用 ViewAnimator 的几个子类，下面一一进行详细介绍。

2.7.1 ViewSwitcher 的功能与用法

ViewSwitcher 代表了视图切换组件，它本身继承了 FrameLayout，因此可以将多个 View 层叠在一起，每次只显示一个组件。当程序控制从一个 View 切换到另一个 View 时，ViewSwitcher 支持指定动画效果。

为了给 ViewSwitcher 添加多个组件，一般通过调用 ViewSwitcher 的 setFactory (ViewSwitcher.ViewFactory)方法为之设置 ViewFactory，并由该 ViewFactory 为之创建 View 即可。

下面通过一个实例来介绍 ViewFactory 的用法。

实例：仿 Android 系统 Launcher 界面

Android 早期版本的 Launcher 界面是上下滚动的，Android 4.2 的 Launcher 界面已经实现了分屏、左右滚动（可能是模仿 iOS 的操作习惯吧），本实例就是通过 ViewSwitcher 来实现 Android 4.2 的分屏、左右滚动效果。

为了实现该效果，程序主界面考虑使用 ViewSwticher 来组合多个 GridView，每个 GridView 代表一个屏幕的应用程序，GridView 中每个单元格显示一个应用程序的图标和程序名。

该程序的主界面布局文件如下。

程序清单：codes\02\2.7\ViewSwitcherTest\res\layout\main.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
        <!-- 定义一个 ViewSwitcher 组件 -->
        <ViewSwitcher
            android:id="@+id/viewSwitcher"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" />
        <!-- 定义滚动到上一屏的按钮 -->
        <Button
            android:id="@+id/button_prev"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentLeft="true"
            android:onClick="prev"
            android:text="&lt;" />
        <!-- 定义滚动到下一屏的按钮 -->
        <Button
            android:id="@+id/button_next"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"
            android:onClick="next"
            android:text="&gt;" />
</RelativeLayout>
```

上面的布局文件中只是定义了一个 ViewSwitcher 组件和两个按钮，这两个按钮分别用于控制该 ViewSwitcher 显示上一屏、下一屏的程序列表。



提示：

这个程序采用了按钮来控制滚动到上一屏、下一屏，这种操作方式显得不够“酷”，实际上完全可以实现通过手势来滚动屏幕。如果希望实现通过手势来滚动屏幕的效果，建议参考本书第 8 章关于手势的知识。

该实例的重点在于为该 ViewSwitcher 设置 ViewFactory 对象，并且当用户单击“<”和“>”两个按钮时控制 ViewSwitcher 显示“上一屏”和“下一屏”的应用程序。

该程序会考虑使用扩展 BaseAdapter 的方式为 GridView 提供 Adapter，而本实例的关键就是根据用户单击的按钮来动态计算该 BaseAdapter 应该显示哪些程序列表。该程序的 Activity 代码如下。

程序清单：codes\02\2.7\ViewSwitcherTest\src\org\crazyit\ui\ViewSwitcherTest.java

```
public class ViewSwitcherTest extends Activity
{
    // 定义一个常量，用于显示每屏显示的应用程序数
    public static final int NUMBER_PER_SCREEN = 12;
    // 代表应用程序的内部类，
    public static class DataItem
    {
        // 应用程序名称
        public String dataName;
        // 应用程序图标
        public Drawable drawable;
    }
    // 保存系统所有应用程序的 List 集合
    private ArrayList<DataItem> items = new ArrayList<DataItem>();
    // 记录当前正在显示第几屏的程序
    private int screenNo = -1;
```

```
// 保存程序所占的总屏数
private int screenCount;
ViewSwitcher switcher;
// 创建 LayoutInflater 对象
LayoutInflater inflater;
@Override
public void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      inflater = LayoutInflater.from(ViewSwitcherTest.this);
      // 创建一个包含 40 个元素的 List 集合，用于模拟包含 40 个应用程序
      for (int i = 0; i < 40; i++)
      {
            String label = "" + i;
            Drawable drawable = getResources().getDrawable(
                        R.drawable.ic_launcher);
            DataItem item = new DataItem();
            item.dataName = label;
            item.drawable = drawable;
            items.add(item);
      }
      // 计算应用程序所占的总屏数
      // 如果应用程序的数量能整除 NUMBER_PER_SCREEN，除法的结果就是总屏数
      // 如果不能整除，总屏数应该是除法的结果再加 1
      screenCount = items.size() % NUMBER_PER_SCREEN == 0 ?
                  items.size()/ NUMBER_PER_SCREEN :
                  items.size() / NUMBER_PER_SCREEN    + 1;
      switcher = (ViewSwitcher) findViewById(R.id.viewSwitcher);
      switcher.setFactory(new ViewFactory()
      {
            // 实际上就是返回一个 GridView 组件
            @Override
            public View makeView()
            {
                  // 加载 R.layout.slidelistview 组件，实际上就是一个 GridView 组件
                  return inflater.inflate(R.layout.slidelistview, null);
            }
      });
      // 页面加载时先显示第一屏
      next(null);
}
public void next(View v)
{
      if (screenNo < screenCount - 1)
      {
            screenNo++;
            // 为 ViewSwitcher 的组件显示过程设置动画
            switcher.setInAnimation(this, R.anim.slide_in_right);
            // 为 ViewSwitcher 的组件隐藏过程设置动画
            switcher.setOutAnimation(this, R.anim.slide_out_left);
            // 控制下一屏将要显示的 GridView 对应的 Adapter
            ((GridView) switcher.getNextView()).setAdapter(adapter);
            // 单击右边按钮，显示下一屏
            // 学习手势检测后，也可通过手势检测实现显示下一屏
            switcher.showNext();  //①
      }
}
public void prev(View v)
{
      if (screenNo > 0)
```

```
            {
                screenNo--;
                // 为 ViewSwitcher 的组件显示过程设置动画
                switcher.setInAnimation(this, android.R.anim.slide_in_left);
                // 为 ViewSwitcher 的组件隐藏过程设置动画
                switcher.setOutAnimation(this, android.R.anim.slide_out_right);
                // 控制下一屏将要显示的 GridView 对应的 Adapter
                ((GridView) switcher.getNextView()).setAdapter(adapter);
                // 单击左边按钮，显示上一屏，当然可以采用手势
                // 学习手势检测后，也可通过手势检测实现显示上一屏
                switcher.showPrevious();  //②
            }
        }
        // 该 BaseAdapter 负责为每屏显示的 GridView 提供列表项
        private BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                // 如果已经到了最后一屏，且应用程序的数量不能整除 NUMBER_PER_SCREEN
                if (screenNo == screenCount - 1
                        && items.size() % NUMBER_PER_SCREEN != 0)
                {
                    // 最后一屏显示的程序数为应用程序的数量对 NUMBER_PER_SCREEN 求余
                    return items.size() % NUMBER_PER_SCREEN;
                }
                // 否则每屏显示的程序数量为 NUMBER_PER_SCREEN
                return NUMBER_PER_SCREEN;
            }
            @Override
            public DataItem getItem(int position)
            {
                // 根据 screenNo 计算第 position 个列表项的数据
                return items.get(screenNo * NUMBER_PER_SCREEN + position);
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            @Override
            public View getView(int position
                    , View convertView, ViewGroup parent)
            {
                View view = convertView;
                if (convertView == null)
                {
                    // 加载 R.layout.labelicon 布局文件
                    view = inflater.inflate(R.layout.labelicon, null);
                }
                // 获取 R.layout.labelicon 布局文件中的 ImageView 组件，并为之设置图标
                ImageView imageView = (ImageView)
                        view.findViewById(R.id.imageview);
                imageView.setImageDrawable(getItem(position).drawable);
                // 获取 R.layout.labelicon 布局文件中的 TextView 组件，并为之设置文本
                TextView textView = (TextView)
                        view.findViewById(R.id.textview);
                textView.setText(getItem(position).dataName);
                return view;
            }
        };
    }
```

上面的程序使用 screenNo 保存当前正在显示第几屏的程序列表。该程序的关键在于粗体字代码部分，该粗体字代码创建了一个 BaseAdapter 对象，这个 BaseAdapter 会根据 screenNo 动态计算该 Adapter 总共包含多少个列表项（如 getCount()方法所示）。会根据 screenNo 计算每个列表项的数据（如 getItem(int position)方法所示）。

BaseAdapter 的 getView()只是简单加载了 R.layout.labelicon 布局文件，并使用当前列表项的图片数据填充 R.layout.labelicon 布局文件中的 ImageView，使用当前列表项的文本数据填充 R.layout.labelicon 布局文件中的 TextView。下面是 R.layout.labelicon 布局文件的代码。

程序清单：codes\02\2.7\ViewSwitcherTest\res\layout\labelicon.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个垂直的 LinearLayout，该容器中放置一个 ImageView 和一个 TextView -->
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:gravity="center">
      <ImageView
          android:id="@+id/imageview"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
             />
      <TextView
          android:id="@+id/textview"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
             />
</LinearLayout>
```

当用户单击“>”按钮时，程序的事件处理函数将会控制 ViewSwitcher 调用 showNext()方法显示下一屏的程序列表——而且此时 screenNo 被加 1，因而 Adapter 将会动态计算下一屏的程序列表，再将该 Adapter 传给 ViewSwitcher 接下来要显示的 GridView。

为了实现 ViewSwitcher 切换 View 时的动画效果，程序的事件处理方法中调用了 ViewSwitcher 的 setInAnimation()、setOutAnimation()方法来设置动画效果。本程序不仅利用了 Android 系统提供的两个动画资源，还自行提供了动画资源。

其中 R.anim.slide_in_right 动画资源对应的代码如下。

程序清单：codes\02\2.7\ViewSwitcherTest\res\anim\slide_in_right.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 设置从右边拖进来的动画
    android:duration 指定动画持续时间  -->
      <translate
            android:fromXDelta="100%p"
            android:toXDelta="0"
            android:duration="@android:integer/config_mediumAnimTime" />
</set>
```

其中 R.anim.slide_out_left 动画资源对应的代码如下。

程序清单：codes\02\2.7\ViewSwitcherTest\res\anim\slide_out_left.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <!-- 设置从左边拖出去的动画
    android:duration 指定动画持续时间 -->
      <translate
            android:fromXDelta="0"
            android:toXDelta="-100%p"
            android:duration="@android:integer/config_mediumAnimTime" />
</set>
```



图 2.57　仿 Android 系统 Launcher 界面

运行上面的程序，可以看到如图 2.57 所示效果。

在图 2.57 所示界面中，当用户单击底端的“<”或“>”按钮时，将可以看到 ViewSwitcher 切换屏幕时的动画效果。

2.7.2　图像切换器（ImageSwitcher）的功能与用法

ImageSwitcher 继承了 ViewSwitcher，因此它具有与 ViewSwitcher 相同的特征：可以在切换 View 组件时使用动画效果。ImageSwitcher 继承了 ViewSwitcher，并重写了 ViewSwitcher 的 showNext()、showPrevious()方法，因此 ImageSwitcher 使用起来更加简单。使用 ImageSwitcher 只要如下两步即可。

（1）为 ImageSwitcher 提供一个 ViewFactory，该 ViewFactory 生成的 View 组件必须是 ImageView。

（2）需要切换图片时，只要调用 ImageSwitcher 的 setImageDrawable(Drawable drawable)、setImageResource(int resid)和 setImageURI(Uri uri)方法更换图片即可。



提示：

ImageSwitcher 与 ImageView 的功能有点相似，它们都可用于显示图片，区别在于 ImageSwitcher 的效果更炫，它可以指定图片切换时的动画效果。

下面通过一个实例来介绍 ImageSwitcher 的用法。

实例：支持动画的图片浏览器

下面的实例是对前面的 GridViewTest 实例的修改，该实例使用 ImageSwitcher 代替了原有的 ImageView。该实例的布局文件如下。

程序清单：codes\02\2.7\ImageSwitcherTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal">
<!-- 定义一个 GridView 组件 -->
<GridView
      android:id="@+id/grid01"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:horizontalSpacing="pt"
      android:verticalSpacing="2pt"
      android:numColumns="4"
      android:gravity="center"/>
<!-- 定义一个 ImageSwitcher 组件 -->
```

```
<ImageSwitcher android:id="@+id/switcher"
      android:layout_width="300dp"
      android:layout_height="300dp"
      android:layout_gravity="center_horizontal"
      android:inAnimation="@android:anim/fade_in"
      android:outAnimation="@android:anim/fade_out"/>
</LinearLayout>
```

上面布局文件的粗体字代码定义了一个 ImageSwitcher，并通过 android:inAnimation 和 android:outAnimation 指定了图片切换时的动画效果。

接下来 Activity 代码需要为该 ImageSwitcher 设置 ViewFactory，并让该 ViewFactory 的 makeView()方法返回 ImageView。下面是该 Activity 的代码。

程序清单：codes\02\2.7\ImageSwitcherTest\src\org\crazyit\ui\ImageSwitcherTest.java

```
public class ImageSwitcherTest extends Activity
{
      int[] imageIds = new int[]
      {
            R.drawable.bomb5 , R.drawable.bomb6 , R.drawable.bomb7
            , R.drawable.bomb8 , R.drawable.bomb9 , R.drawable.bomb10
            , R.drawable.bomb11 , R.drawable.bomb12       , R.drawable.bomb13
            , R.drawable.bomb14 , R.drawable.bomb15 , R.drawable.bomb16
      };
      ImageSwitcher switcher;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 创建一个 List 对象，List 对象的元素是 Map
            List<Map<String, Object>> listItems =
                        new ArrayList<Map<String, Object>>();
            for (int i = 0; i < imageIds.length; i++)
            {
                  Map<String, Object> listItem = new HashMap<String, Object>();
                  listItem.put("image", imageIds[i]);
                  listItems.add(listItem);
            }
            // 获取显示图片的 ImageSwitcher
            switcher = (ImageSwitcher)
                        findViewById(R.id.switcher);
            // 为 ImageSwitcher 设置图片切换的动画效果
            switcher.setFactory(new ViewFactory()
            {
                  @Override
                  public View makeView()
                  {
                        // 创建 ImageView 对象
                        ImageView imageView = new ImageView(ImageSwitcherTest.this);
                        imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
                        imageView.setLayoutParams(new ImageSwitcher.LayoutParams(
                              LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
                        // 返回 ImageView 对象
                        return imageView;
                  }
            });
            // 创建一个 SimpleAdapter
            SimpleAdapter simpleAdapter = new SimpleAdapter(this,
                  listItems
                  // 使用/layout/cell.xml 文件作为界面布局
```

```
                , R.layout.cell, new String[]{"image"},
                new int[] { R.id.image1 });
        GridView grid = (GridView) findViewById(R.id.grid01);
        // 为 GridView 设置 Adapter
        grid.setAdapter(simpleAdapter);
        // 添加列表项被选中的监听器
        grid.setOnItemSelectedListener(new OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view,
                    int position, long id)
            {
                // 显示当前被选中的图片
                switcher.setImageResource(imageIds[position]);
            }
            @Override
            public void onNothingSelected(AdapterView<?> parent)
            {
            }
        });
        // 添加列表项被单击的监听器
        grid.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id)
            {
                // 显示被单击的图片
                switcher.setImageResource(imageIds[position]);
            }
        });
    }
}
```



图 2.58　ImageSwitcher 测试

上面的代码中第一段粗体字代码重写了 ViewFactory 的 makeView()方法，该方法返回一个 ImageView 对象，这样该 ImageSwitcher 即可正常工作。该 Activity 还为 GridView 绑定了事件监听器，当用户单击 GridView 或选中 GridView 的指定单元格时，ImageSwitcher 切换为显示对应的图片。

运行上面的实例，可以看到如图 2.58 所示界面。

在图 2.58 所示界面中，用户单击或选中 GridView 中某个图标时，下面的 ImageSwitcher 将会切换为显示对应的图片，图片切换时会使用动画效果。

2.7.3　文本切换器（TextSwitcher）的功能与用法

TextSwitcher 继承了 ViewSwitcher，因此它具有与 ViewSwitcher 相同的特征：可以在切换 View 组件时使用动画效果。与 ImageSwitcher 相似的是，使用 TextSwitcher 也需要设置一个 ViewFactory。与 ImageSwitcher 不同的是，TextSwitcher 所需的 ViewFactory 的 makeView()方法必须返回一个 TextView 组件。



提示：

TextSwitcher 与 TextView 的功能有点相似，它们都可用于显示文本内容，区别在于 TextSwitcher 的效果更炫，它可以指定文本切换时的动画效果。

下面的实例在界面上定义了一个 TextSwitcher，界面布局文件如下。

程序清单：codes\02\2.7\TextSwitcherTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      android:layout_width="match_parent"
      android:layout_height="match_parent" >
<!-- 定义一个 TextSwitcher，并指定了文本切换时的动画效果 -->
<TextSwitcher
    android:id="@+id/textSwitcher"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:inAnimation="@android:anim/slide_in_left"
      android:outAnimation="@android:anim/slide_out_right"
      android:onClick="next"/>
</LinearLayout>
```

上面的布局文件中定义了一个 TextSwitcher，并为该文本切换指定了文本切换时的动画效果。接下来 Activity 只要为该 TextSwitcher 设置 ViewFactory，该 TextSwitcher 即可正常工作。

下面是该 Activity 的代码。

程序清单：codes\02\2.7\TextSwitcherTest\src\org\crazyit\ui\TextSwitcherTest.java

```
public class TextSwitcherTest extends Activity
{
      TextSwitcher textSwitcher;
      String[] strs = new String[]
            {
                  "疯狂 Java 讲义",
                  "轻量级 Java EE 企业应用实战",
                  "疯狂 Android 讲义",
                  "疯狂 Ajax 讲义"
            };
      int curStr;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            textSwitcher = (TextSwitcher) findViewById(R.id.textSwitcher);
            textSwitcher.setFactory(new ViewSwitcher.ViewFactory()
            {
                  public View makeView()
                  {
                        TextView tv = new TextView(TextSwitcherTest.this);
                        tv.setTextSize(40);
                        tv.setTextColor(Color.MAGENTA);
                        return tv;
                  }
            });
            // 调用 next 方法显示下一个字符串
            next(null);
      }
      // 事件处理函数，控制显示下一个字符串
      public void next(View source)
      {
            textSwitcher.setText(strs[curStr++ % strs.length]);  //①
      }
}
```

上面的代码重写了 ViewFactory 的 makeView()方法，该方法返回了一个 TextView，这样即可让 TextSwitcher 正常工作。当程序要切换 TextSwitcher 显示文本时，调用 TextSwitcher 的 setText()方法修改文本即可——如上面的①号代码所示。



图 2.59　文本切换器

运行上面的程序，可以看到如图 2.59 所示界面。

单击图 2.59 所示界面上的文本切换器，界面将会显示下一个文本，文本切换时将会使用动画效果。

2.7.4　ViewFlipper 的功能与用法

ViewFlipper 组件继承了 ViewAnimator，它可调用 addView(View v)添加多个组件，一旦向 ViewFlipper 中添加了多个组件之后，ViewFlipper 可使用动画控制多个组件之间的切换效果。

ViewFlipper 与前面介绍的 AdapterViewFlipper 有较大的相似性，它们可以控制组件切换的动画效果。它们的区别是：ViewFlipper 需要开发者通过 addView(View v)添加多个 View，而 AdapterViewFlipper 则只要传入一个 Adapter，Adapter 将会负责提供多个 View。因此 ViewFlipper 可以指定与 AdapterViewFlipper 相同的 XML 属性。

实例：自动播放的图片库

该实例与前面介绍的 AdapterViewFlipper 实例非常相似，区别只是该实例直接定义了该 ViewFlipper 所包含的 View 组件。下面是该实例的界面布局文件。

程序清单：codes\02\2.7\ViewFlipperTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
      <ViewFlipper
            android:id="@+id/details"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:flipInterval="1000">
            <ImageView
                  android:src="@drawable/java"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
            </ImageView>
            <ImageView
                  android:src="@drawable/android"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
            </ImageView>
            <ImageView
                  android:src="@drawable/ee"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
            </ImageView>
      </ViewFlipper>
      <Button
            android:text="&lt;"
            android:onClick="prev"
            android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentLeft="true"/>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_centerInParent="true"
            android:onClick="auto"
            android:text="自动播放"/>
        <Button
            android:text="&gt;"
            android:onClick="next"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"/>
</RelativeLayout>
```

上面的布局文件定义了一个 ViewFlipper，并在该 ViewFlipper 中定义了三个 ImageView，这意味着该 ViewFlipper 包含了三个子组件。接下来在 Activity 代码中即可调用 ViewFlipper 的 showPrevious()、showNext()等方法控制 ViewFlipper 显示上一个、下一个子组件。为了控制组件切换时的动画效果，还需要调用 ViewFlipper 的 setInAnimation()、setOutAnimation()方法设置动画效果。

下面是该 Activity 的代码。

程序清单：codes\02\2.7\ViewFlipperTest\src\org\crazyit\ui\ViewFlipperTest.java

```
public class ViewFlipperTest extends Activity
{
    private ViewFlipper viewFlipper;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        viewFlipper = (ViewFlipper) findViewById(R.id.details);
    }
    public void prev(View source)
    {
        viewFlipper.setInAnimation(this , R.anim.slide_in_right);
        viewFlipper.setOutAnimation(this , R.anim.slide_out_left);
        // 显示上一个组件
        viewFlipper.showPrevious();
        // 停止自动播放
        viewFlipper.stopFlipping();
    }
    public void next(View source)
    {
        viewFlipper.setInAnimation(this , android.R.anim.slide_in_left);
        viewFlipper.setOutAnimation(this , android.R.anim.slide_out_right);
        // 显示下一个组件
        viewFlipper.showNext();
        // 停止自动播放
        viewFlipper.stopFlipping();
    }
    public void auto(View source)
    {
        viewFlipper.setInAnimation(this , android.R.anim.slide_in_left);
        viewFlipper.setOutAnimation(this , android.R.anim.slide_out_right);
```

```
            // 开始自动播放
            viewFlipper.startFlipping();
        }
    }
```

上面的程序中粗体字代码就是控制 ViewFlipper 切换组件的动画效果，以及控制 ViewFlipper 切换组件的关键代码。运行该程序，可以看到如图 2.60 所示界面。



图 2.60　ViewFlipper 控制图片切换

单击图 2.60 所示界面下方的按钮，即可看到图片上、下切换，而且组件切换时将可以看到动画切换效果。

2.8　各种杂项组件

除了前面介绍的 6 组 UI 组件之外，Android 还有如下一些常用的杂项组件，掌握这些杂项组件也是开发 Android 应用必需的技能。

2.8.1　使用 Toast 显示提示信息框

Toast 是一种非常方便的提示消息框，它会在程序界面上显示一个简单的提示信息。这个提示信息框用于向用户生成简单的提示信息。它具有如下两个特点。

- Toast 提示信息不会获得焦点。
- Toast 提示信息过一段时间会自动消失。

使用 Toast 来生成提示消息也非常简单，只要如下几个步骤即可。

1. 调用 Toast 的构造器或 makeText()静态方法创建一个 Toast 对象。
2. 调用 Toast 的方法来设置该消息提示的对齐方式、页边距等。
3. 调用 Toast 的 show()方法将它显示出来。

Toast 的功能和用法都比较简单，大部分时候它只能显示简单的文本提示；如果应用需要显示诸如图片、列表之类的复杂提示，一般建议使用对话框来完成；如果开发者确实想通过 Toast 来完成，也是可以的，此时就需要调用 Toast 构造器创建实例，再调用 setView()方法设置该 Toast 显示的 View 组件。该方法允许开发者自己定义 Toast 显示的内容。

下面以一个示例程序来示范 Toast 的用法。

实例：带图片的消息提示

本示例程序非常简单，它在用户界面上显示了两个按钮，其中一个按钮用于激发普通的 Toast 提示，另一个按钮用于激发带图片的 Toast 提示。这意味着开发者必须调用该 Toast 对象的 setView()方法来改变该 Toast 对象的内容 View。

本程序的用户界面很简单，只有两个普通按钮，故不再给出界面布局文件代码。本程序的 Java 代码如下。

程序清单：codes\02\2.8\ToastTest\src\org\crazyit\ui\ToastTest.java

```
public class ToastTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button simple = (Button) findViewById(R.id.simple);
        // 为按钮的单击事件绑定事件监听器
        simple.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建一个 Toast 提示信息
                Toast toast = Toast.makeText(ToastTest.this
                    , "简单的提示信息"
                    // 设置该 Toast 提示信息的持续时间
                    , Toast.LENGTH_SHORT);
                toast.show();
            }
        });
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮的单击事件绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建一个 Toast 提示信息
                Toast toast = new Toast(ToastTest.this);
                // 设置 Toast 的显示位置
                toast.setGravity(Gravity.CENTER, 0, 0);
                // 创建一个 ImageView
                ImageView image = new ImageView(ToastTest.this);
                image.setImageResource(R.drawable.tools);
                // 创建一个 LinearLayout 容器
                LinearLayout ll = new LinearLayout(ToastTest.this);
                // 向 LinearLayout 中添加图片、原有的 View
                ll.addView(image);
                // 创建一个 ImageView
                TextView textView = new TextView(ToastTest.this);
                textView.setText("带图片的提示信");
                // 设置文本框内字号的大小和字体颜色
                textView.setTextSize(30);
                textView.setTextColor(Color.MAGENTA);
                ll.addView(textView);
                // 设置 Toast 显示自定义 View
                toast.setView(ll);
                // 设置 Toast 的显示时间
                toast.setDuration(Toast.LENGTH_LONG);
                toast.show();
            }
        });
    }
}
```

上面的程序比较简单：第一个按钮被单击时，程序只是简单地创建了一个 Toast 对象，并把它显示出来，因此单击第一个按钮只是看到一个简单的 Toast 提示；当第二个按钮被单击时，程序先创建了一个 Toast 对象，并调用该 Toast 对象的 setView()方法改变了该消息提示的内容。因此单击第二个按钮时将看到带图片的消息提示，如图 2.61 所示。



图 2.61 带图片的消息提示

2.8.2 日历视图（CalendarView）组件的功能和用法

日历视图（CalendarView）可用于显示和选择日期，用户既可选择一个日期，也可通过触摸来滚动日历。如果希望监控该组件的日期改变，可调用 CalendarView 的 setOnDateChangeListener()方法为此组件的点击事件添加事件监听器。

使用 CalendarView 时可指定如表 2.30 所示的 XML 属性。

表 2.30　CalendarView 支持的常见 XML 属性

XML 属性	相关方法	说　明
android:dateTextAppearance	setDateTextAppearance(int)	设置该日历视图的日期文字的样式
android:firstDayOfWeek	setFirstDayOfWeek(int)	设置每周的第一天，允许设置周一到周日任意一天作为每周的第一天
android:focusedMonthDateColor	setFocusedMonthDateColor(int)	设置获得焦点的月份的日期文字的颜色
android:maxDate	setMaxDate(long)	设置该日历组件支持的最大日期。以 mm/dd/yyyy 格式指定最大日期
android:minDate	setMinDate(long)	设置该日历组件支持的最小日期。以 mm/dd/yyyy 格式指定最小日期
android:selectedDateVerticalBar	setSelectedDateVerticalBar(int)	设置绘制在选中日期两边的竖线对应的 Drawable
android:selectedWeekBackgroundColor	setSelectedWeekBackgroundColor(int)	设置被选中周的背景色
android:showWeekNumber	setShowWeekNumber(boolean)	设置是否显示第几周
android:shownWeekCount	setShownWeekCount(int)	设置该日历组件总共显示几个星期
android:unfocusedMonthDateColor	setUnfocusedMonthDateColor(int)	设置没有焦点的月份的日期文字的颜色
android:weekDayTextAppearance	setWeekDayTextAppearance(int)	设置星期几的文字样式
android:weekNumberColor	setWeekNumberColor(int)	设置显示周编号的颜色
android:weekSeparatorLineColor	setWeekSeparatorLineColor(int)	设置周分割线的颜色

下面通过实例来示范 CalendarView 组件的功能与用法。

实例：选择您的生日

下面实例的重点在于布局文件中添加了一个 CalendarView 组件。下面是该实例的布局文件。

程序清单：codes\02\2.8\CalendarViewTest\res\layout\main.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
       android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="选择您的生日："
       />
    <!-- 设置以星期二作为每周第一天
    设置该组件总共显示 4 个星期
    并对该组件的日期时间进行了定制 -->
```

```
        <CalendarView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:firstDayOfWeek="3"
            android:shownWeekCount="4"
            android:selectedWeekBackgroundColor="#aff"
            android:focusedMonthDateColor="#f00"
            android:weekSeparatorLineColor="#ff0"
            android:unfocusedMonthDateColor="#f9f"
            android:id="@+id/calendarView" />
    </LinearLayout>
```

上面的布局文件中粗体字代码定义了一个 CalenarView 组件，并设置该组件总共只显示 4 周，以每周的星期二作为第一天。

为了监听用户选择日期的事件，本实例在 Activity 代码中调用该组件的 setOnDateChangeListener()方法来添加事件监听器。该实例的 Activity 代码如下。

程序清单：codes\02\2.8\CalendarViewTest\src\org\crazyit\ui\CalendarViewTest.java

```
public class CalendarViewTest extends Activity
{
    CalendarView cv;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        cv = (CalendarView)findViewById(R.id.calendarView);
        // 为 CalendarView 组件的日期改变事件添加事件监听器
        cv.setOnDateChangeListener(new OnDateChangeListener()
        {
            @Override
            public void onSelectedDayChange(CalendarView view, int year,
                    int month, int dayOfMonth)
            {
                // 使用 Toast 显示用户选择的日期
                Toast.makeText(CalendarViewTest.this,
                    "你生日是" + year + "年" + month + "月"
                    + dayOfMonth + "日" ,
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

上面 Activity 的粗体字代码实现了 onSelectedDayChange()，当用户选择的日期发生改变时将会激发该方法。运行上面的程序，当用户选择的日期发生改变时，将可以看到程序显示如图 2.62 所示界面。



图 2.62 通过 CalenarView 组件选择日期

➤➤2.8.3 日期、时间选择器（DatePicker 和 TimePicker）的功能和用法

DatePicker 和 TimePicker 是两个比较易用的控件，它们都从 FrameLayout 派生而来，其中 DatePicker 供用户选择日期；而

TimePicker 则供用户选择时间。

DatePicker 和 TimePicker 在 FrameLayout 的基础上提供了一些方法来获取当前用户所选择的日期、时间；如果程序需要获取用户选择的日期、时间，则可通过为 DatePicker 添加 OnDateChangedListener 进行监听、为 TimePicker 添加 OnTimerChangedListener 进行监听来实现。

使用 DatePicker 时可指定如表 2.31 所示的 XML 属性。

表 2.31　DatePicker 支持的常见 XML 属性

XML 属性	说　明
android:calendarViewShown	设置该日期选择是否显示 CalendarView 组件
android:endYear	设置日期选择器允许选择的最后一年
android:maxDate	设置该日期选择器的最大日期。以 mm/dd/yyyy 格式指定最大日期
android:minDate	设置该日历组件支持的最小日期。以 mm/dd/yyyy 格式指定最小日期
android:spinnersShown	设置该日期选择器是否显示 Spinner 日期选择组件
android:startYear	设置日期选择器允许选择的第一年

下面以一个让用户选择日期、时间的实例来示范 DatePicker 和 TimePicker 的功能和用法。

实例：用户选择日期、时间

为了让用户能选择日期，本应用需要同时使用 DatePicker 和 TimePicker 两个组件，并为它们分别绑定监听器。下面是本应用的界面布局。

程序清单：codes\02\2.8\ChooseDate\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="选择购买本书的具体时间"
      />
<!-- 定义一个 DatePicker 组件 -->
<DatePicker android:id="@+id/datePicker"
      android:layout_width="wrap_content"
      android:layout_height="200dp"
      android:layout_gravity="center_horizontal"
      android:startYear="2000"
      android:endYear="2012"
      android:calendarViewShown="true"
      android:spinnersShown="true"
      />
<!-- 定义一个 TimePicker 组件 -->
<TimePicker android:id="@+id/timePicker"
      android:layout_width="wrap_content"
      android:layout_height="100dp"
      android:layout_gravity="center_horizontal"
      />
<!-- 显示用户输入日期、时间的控件 -->
<EditText android:id="@+id/show"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:editable="false"
        android:cursorVisible="false"
        />
</LinearLayout>
```

上面的界面布局中添加了一个 DatePicker、一个 TimePicker，这两个组件供用户选择日期、时间。除此之外，上面的界面布局中还包含一个 EditText，该组件用于显示用户选择的日期、时间。

主程序如下。

程序清单：codes\02\2.8\ChooseDate\src\org\crazyit\ui\ChooseDate.java

```
public class ChooseDate extends Activity
{
    // 定义 5 个记录当前时间的变量
    private int year;
    private int month;
    private int day;
    private int hour;
    private int minute;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        DatePicker datePicker = (DatePicker)
            findViewById(R.id.datePicker);
        TimePicker timePicker = (TimePicker)
            findViewById(R.id.timePicker);
        // 获取当前的年、月、日、小时、分钟
        Calendar c = Calendar.getInstance();
        year = c.get(Calendar.YEAR);
        month = c.get(Calendar.MONTH);
        day = c.get(Calendar.DAY_OF_MONTH);
        hour = c.get(Calendar.HOUR);
        minute = c.get(Calendar.MINUTE);
        // 初始化 DatePicker 组件，初始化时指定监听器
        datePicker.init(year, month, day, new OnDateChangedListener()
        {
            @Override
            public void onDateChanged(DatePicker arg0, int year
                    , int month, int day)
            {
                ChooseDate.this.year = year;
                ChooseDate.this.month = month;
                ChooseDate.this.day = day;
                // 显示当前日期、时间
                showDate(year, month, day, hour, minute);
            }
        });
        // 为 TimePicker 指定监听器
        timePicker.setOnTimeChangedListener(new OnTimeChangedListener()
        {
            @Override
            public void onTimeChanged(TimePicker view
                    , int hourOfDay, int minute)
            {
                ChooseDate.this.hour = hourOfDay;
                ChooseDate.this.minute = minute;
```

```
                        // 显示当前日期、时间
                        showDate(year, month, day, hour, minute);
                    }
                });
        }
        // 定义在 EditText 中显示当前日期、时间的方法
        private void showDate(int year, int month
                , int day, int hour, int minute)
        {
            EditText show = (EditText) findViewById(R.id.show);
            show.setText("您的购买日期为：" + year + "年"
                    + (month + 1) + "月" + day + "日 "
                    + hour + "时" + minute + "分");
        }
}
```

上面的程序中两行粗体字代码就是分别为 DatePicker、TimePicker 绑定事件监听器的代码，DatePicker 和 TimePicker 绑定监听器的方式略有不同，但本质还是一样的。一旦为 DatePicker、TimePicker 绑定了监听器，当用户通过这两个组件来选择日期、时间时，监听器被触发——监听器负责使用 EditText 来显示用户选择的日期、时间。运行上面的程序将看到如图 2.63 所示界面。



图 2.63　选择日期、时间

➤➤2.8.4　数值选择器（NumberPicker）的功能与用法

数值选择器用于让用户输入数值，用户既可以通过键盘输入数值，也可以通过拖动来选择数值。使用该组件常用如下三个方法。

- setMinValue(int minVal)：设置该组件支持的最小值。
- setMaxValue(int maxVal)：设置该组件支持的最大值。
- setValue(int value)：设置该组件的当前值。

下面通过一个实例来介绍 NumberPicker 的功能与用法。

实例：选择您意向的价格范围

在该实例中，程序将会使用两个 NumberPicker 来让用户选择价格，第一个 NumberPicker 用于选择低价，第二个 NumberPicker 用于选择高价。下面是该实例的布局文件。

程序清单：codes\02\2.8\NumberPickerTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TableRow
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView
            android:text="选择低价："
            android:layout_width="120dp"
            android:layout_height="wrap_content" />
        <NumberPicker
```

```
            android:id="@+id/np1"
            android:layout_width="match_parent"
            android:layout_height="80dp"
            android:focusable="true"
            android:focusableInTouchMode="true" />
    </TableRow>
    <TableRow
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView
            android:text="选择高价："
            android:layout_width="120dp"
            android:layout_height="wrap_content" />
        <NumberPicker
            android:id="@+id/np2"
            android:layout_width="match_parent"
            android:layout_height="80dp"
            android:focusable="true"
            android:focusableInTouchMode="true" />
    </TableRow>
</TableLayout>
```

上面的布局文件中定义了两个 NumberPicker，接下来 Activity 代码需要为这两个 NumberPicker 设置最小值、最大值，并为它们绑定事件监听器。下面是该 Activity 的代码。

程序清单：codes\02\2.8\NumberPickerTest\org\crazyit\ui\NumberPickerTest.java

```
public class NumberPickerTest extends Activity
{
    NumberPicker np1, np2;
    // 定义最低价格、最高价格的初始值
    int minPrice = 25, maxPrice = 75;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        np1 = (NumberPicker) findViewById(R.id.np1);
        // 设置 np1 的最小值和最大值
        np1.setMinValue(10);
        np1.setMaxValue(50);
        // 设置 np1 的当前值
        np1.setValue(minPrice);
        np1.setOnValueChangedListener(new OnValueChangeListener()
        {
            // 当 NumberPicker 的值发生改变时，将会激发该方法
            @Override
            public void onValueChange(NumberPicker picker, int oldVal,
                    int newVal)
            {
                minPrice = newVal;
                showSelectedPrice();
            }
        });
        np2 = (NumberPicker) findViewById(R.id.np2);
        // 设置 np2 的最小值和最大值
        np2.setMinValue(60);
        np2.setMaxValue(100);
        // 设置 np2 的当前值
        np2.setValue(maxPrice);
        np2.setOnValueChangedListener(new OnValueChangeListener()
        {
```

```
                // 当 NumberPicker 的值发生改变时，将会激发该方法
                @Override
                public void onValueChange(NumberPicker picker, int oldVal,
                        int newVal)
                {
                    maxPrice = newVal;
                    showSelectedPrice();
                }
            });
        }
        private void showSelectedPrice()
        {
            Toast.makeText(this, "您选择最低价格为：" + minPrice
                + ",最高价格为：" + maxPrice, Toast.LENGTH_SHORT)
                .show();
        }
}
```



图 2.64　使用 NumberPicker 选择数值

上面两段粗体字代码的控制逻辑基本是相似的，它们都调用了 NumberPicker 的 setMinValue()、setMaxValue()、setValue()来设置该数值选择器的最小值、最大值和当前值。除此之外，程序还为两个日期选择器绑定了事件监听器：当它们的值发生改变时，将会激发相应的事件处理方法。

运行该程序，并通过 NumberPicker 选择数值，将可以看到如图 2.64 所示界面。

➤➤2.8.5　搜索框（SearchView）的功能与用法

SearchView 是搜索框组件，它可以让用户在文本框内输入文字，并允许通过监听器监控用户输入，当用户输入完成后提交搜索时，也可通过监听器执行实际的搜索。

使用 SearchView 时可使用如下常用方法。

- setIconifiedByDefault (boolean iconified)：设置该搜索框默认是否自动缩小为图标。
- setSubmitButtonEnabled(boolean enabled)：设置是否显示搜索按钮。
- setQueryHint(CharSequence hint)：设置搜索框内默认显示的提示文本。
- setOnQueryTextListener (SearchView.OnQueryTextListener listener)：为该搜索框设置事件监听器。

如果为 SearchView 增加一个配套的 ListView，则可以为 SearchView 增加自动完成的功能。如下实例示范了 SearchView 的功能与用法。

实例：搜索

该实例的界面布局文件中定义了一个 SearchView 和 ListView，其中 ListView 用于为 SearchView 显示自动补齐列表。界面布局文件如下。

程序清单：codes\02\2.8\SearchViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
```

```
        <!-定义一个 SearchView -->
        <SearchView
              android:id="@+id/sv"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content" />
        <!-- 为 SearchView 定义自动完成的 ListView-->
        <ListView
              android:id="@+id/lv"
              android:layout_width="match_parent"
              android:layout_height="0dp"
              android:layout_weight="1"/>
    </LinearLayout>
```

上面的布局文件中定义了一个 SearchView 组件，并为该 SearchView 组件定义了一个 ListView 组件，该 ListView 组件用于为 SearchView 组件显示自动完成列表。

下面是该实例对应的 Activity 代码。

程序清单：codes\02\2.8\SearchViewTest\src\org\crazyit\ui\SearchViewTest.java

```
public class SearchViewTest extends Activity implements
        SearchView.OnQueryTextListener
{
    private SearchView sv;
    private ListView lv;
    // 自动完成的列表
    private final String[] mStrings = { "aaaaa", "bbbbbb", "cccccc" };
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        lv = (ListView) findViewById(R.id.lv);
        lv.setAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1, mStrings));
        lv.setTextFilterEnabled(true);
        sv = (SearchView) findViewById(R.id.sv);
        // 设置该 SearchView 默认是否自动缩小为图标
        sv.setIconifiedByDefault(false);
        // 为该 SearchView 组件设置事件监听器
        sv.setOnQueryTextListener(this);
        // 设置该 SearchView 显示搜索按钮
        sv.setSubmitButtonEnabled(true);
        // 设置该 SearchView 内默认显示的提示文本
        sv.setQueryHint("查找");
    }
    // 用户输入字符时激发该方法
    @Override
    public boolean onQueryTextChange(String newText)
    {
        if (TextUtils.isEmpty(newText))
        {
            // 清除 ListView 的过滤
            lv.clearTextFilter();
        }
        else
        {
            // 使用用户输入的内容对 ListView 的列表项进行过滤
            lv.setFilterText(newText);
        }
        return true;
    }
```

```
        // 单击搜索按钮时激发该方法
        @Override
        public boolean onQueryTextSubmit(String query)
        {
            // 实际应用中应该在该方法内执行实际查询
            // 此处仅使用 Toast 显示用户输入的查询内容
            Toast.makeText(this, "您的选择是:" + query
                    , Toast.LENGTH_SHORT).show();
            return false;
        }
    }
```



图 2.65　使用 SearchView 执行搜索

上面的程序中粗体字代码就是控制 SearchView 的关键代码，第一段粗体字代码为 SeachView 设置了事件监听器，并为该 SearchView 启用了搜索按钮。接下来程序重写了 onQueryTextChange()、onQueryTextSubmit()两个方法，这两个方法用于为 SearchView 的事件提供响应。

运行该程序，将可以看到如图 2.65 所示界面。

2.8.6　选项卡（TabHost）的功能和用法

TabHost 是一种非常实用的组件，TabHost 可以很方便地在窗口上放置多个标签页，每个标签页相当于获得了一个与外部容器相同大小的组件摆放区域。通过这种方式，就可以在一个容器里放置更多组件，例如许多手机系统都会在同一个窗口定义多个标签页来显示通话记录，包括“未接电话”、“已接电话”、“呼出电话”等。

与 TabHost 结合使用的还有如下组件。

- **TabWidget**：代表选项卡的标签条。
- **TabSpec**：代表选项卡的一个 Tab 页面。

TabHost 仅仅是一个简单的容器，它提供了如下两个方法来创建、添加选项卡。

- **newTabSpec(String tag)**：创建选项卡。
- **addTab(TabHost.TabSpec tabSpec)**：添加选项卡。

使用 TabHost 的一般步骤如下。

1. 在界面布局中定义 TabHost 组件，并为该组件定义该选项卡的内容。
2. Activity 应该继承 TabActivity。
3. 调用 TabActivity 的 getTabHost()方法获取 TabHost 对象。
4. 通过 TabHost 对象的方法来创建、添加选项卡。

除此之外，TabHost 还提供了一些方法获取当前选项卡，获取当前 View 的方法，具体可以参考 API 文档。如果程序需要监控 TabHost 里当前标签页的改变，可以为它设置 TabHost.OnTabChangeListener 监听器。

下面通过一个简单的实例来示范选项卡的用法。

实例：通话记录界面

下面的示例程序使用 TabHost 定义一个标签容器，并使用了三个 LinearLayout 来定义标签页（实际上可以使用任何 View 组件来定义标签页）。该程序的界面布局文件如下。

程序清单：codes\02\2.8\TabHostTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <!-- 定义第一个标签页的内容 -->
            <LinearLayout
                android:id="@+id/tab01"
                android:orientation="vertical"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent">
                <!-- 省略两个 TextView 的定义 -->
                ...
            </LinearLayout>
            <!-- 定义第二个标签页的内容 -->
            <LinearLayout
                android:id="@+id/tab02"
                android:orientation="vertical"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent">
                <!-- 省略两个 TextView 的定义 -->
                ...
            </LinearLayout>
            <!-- 定义第三个标签页的内容 -->
            <LinearLayout
                android:id="@+id/tab03"
                android:orientation="vertical"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:textSize="11pt">
                <!-- 省略 2 个 TextView 的定义 -->
                ...
            </LinearLayout>
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

请注意上面的布局文件中粗体字代码部分，从上面的布局文件可以发现，TabHost 容器内部需要组合两个组件：TabWidget 和 FrameLayout，其中 TabWidget 定义选项卡的标题条；FrameLayout 则用于“层叠”组合多个选项页面。不仅如此，上面的布局文件中这三个组件的 ID 也有要求。

- TabHost 的 ID 应该为@android:id/tabhos。
- TabWidget 的 ID 应该为@android:id/tabs。
- FrameLayout 的 ID 应该为@android:id/tabcontent。

上面这三个 ID 并不是开发者自己定义的，而是引用了 Android 系统已有的 ID。

接下来主程序即可加载该布局资源，并将布局文件中的三个 Tab 页面添加到该 TabHost 容器中。该 Activity 代码如下。

程序清单：codes\02\2.8\TabHostTest\src\org\crazyit\ui\TabHostTest.java

```
public class TabHostTest extends TabActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取该 Activity 里面的 TabHost 组件
        TabHost tabHost = getTabHost();
        // 创建第一个 Tab 页
        TabSpec tab1 = tabHost.newTabSpec("tab1")
            .setIndicator("已接电话") // 设置标题
            .setContent(R.id.tab01); //设置内容
        // 添加第一个标签页
        tabHost.addTab(tab1);
        TabSpec tab2 = tabHost.newTabSpec("tab2")
            // 在标签标题上放置图标
            .setIndicator("呼出电话", getResources()
            .getDrawable(R.drawable.ic_launcher))
            .setContent(R.id.tab02);
        // 添加第二个标签页
        tabHost.addTab(tab2);
        TabSpec tab3 = tabHost.newTabSpec("tab3")
            .setIndicator("未接电话")
            .setContent(R.id.tab03);
        // 添加第三个标签页
        tabHost.addTab(tab3);
    }
}
```

上面的程序中粗体字代码就是为 TabHost 创建、并添加 Tab 页面的代码，上面的程序一共添加了三个标签页，因此类似粗体字的代码一共写了三次。其中第二个标签的标题上还添加了一个图片。



图 2.66　通话记录界面

运行上面的程序将看到如图 2.66 所示界面。

上面的程序调用了 TabHost.TabSpec 对象的 setContent(int viewId) 方法来设置标签页内容；除此之外还可调用 setContent(Intent intent) 方法来设置标签页内容，Intent 还可用于启动其他 Activity——这意味着 TabHost.TabSpec 可直接装载另一个 Activity。本书后面有关于 Intent 的详细介绍。

注意：

最新版本的 Android 平台已经不再推荐使用 TabActivity，而是推荐使用 Fragment 来代替 TabActivity，本书第 4 章会详细介绍 Fragment 的相关知识。



2.8.7　滚动视图（ScrollView）的功能和用法

滚动视图 ScrollView 由 FrameLayout 派生而出，它就是一个用于为普通组件添加滚动条

的组件。ScrollView 里最多只能包含一个组件，而 ScrollView 的作用就是为该组件添加垂直滚动条。



提示：

ScrollView 的作用和 Swing 编程中的 JScrollPane 非常相似，它们甚至不能被称为真正的容器。它们只是为其他容器添加滚动条。

默认情况下，ScrollView 只是为其他组件添加垂直滚动条，如果应用需要添加水平滚动条，则可借助于另一个滚动视图——HorizontalScrollView 来实现。ScrollView 与 HorizontalScrollView 的功能基本相似，只是前者添加垂直滚动条，后者添加水平滚动条。

下面以一个例子来示范 ScrollView、HorizontalScrollView 的用法。

实例：可垂直和水平滚动的视图

下面的程序通过在 ScrollView 里嵌套 HorizontalScrollView，来为应用的界面同时添加水平滚动条、垂直滚动条。下面是该应用的界面布局文件。

程序清单：codes\02\2.8\ScrollViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义 ScrollView，为里面的组件添加垂直滚动条 -->
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<!-- 定义 HorizontalScrollView，为里面的组件添加水平滚动条 -->
<HorizontalScrollView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
<LinearLayout android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<!-- 省略多个 TextView 组件 -->
...
</LinearLayout>
</HorizontalScrollView>
</ScrollView>
```

上面的界面布局实现了界面的垂直、水平同时滚动，使用 Activity 显示上面的界面布局，将看到如图 2.67 所示界面。

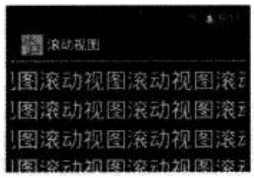


图 2.67 水平、垂直滚动

2.8.8 Notification 的功能与用法

Notification 是显示在手机状态栏的通知——手机状态栏位于手机屏幕的最上方，那里一般显示了手机当前的网络状态、电池状态、时间等。Notification 所代表的是一种具有全局效果的通知，程序一般通过 NotificationManager 服务来发送 Notification。



提示：

NotificationManager 是一个重要的系统服务，该 API 位于图 1.1 中的应用程序框架层，应用程序可通过 NotificationManager 向系统发送全局通知。

Android 3.0 增加 Notification.Builder 类，通过该类允许开发者更轻松地创建 Notification 对象。Notification.Builder 提供了如下常用方法。

- setDefaults()：设置通知 LED 灯、音乐、振动等。
- setAutoCancel()：设置点击通知后，状态栏自动删除通知。
- setContentTitle()：方法设置通知标题。
- setContentText()：设置通知内容。
- setSmallIcon()：为通知设置图标。
- setLargeIcon()：为通知设置大图标。
- setTick()：设置通知在状态栏的提示文本。
- setContentIntent()：设置点击通知后将要启动的程序组件对应的 PendingIntent。

发送 Notification 很简单，按如下步骤进行即可。

① 调用 getSystemService(NOTIFICATION_SERVICE) 方法获取系统的 NotificationManager 服务。

② 通过构造器创建一个 Notification 对象。

③ 为 Notification 设置各种属性。

④ 通过 NotificationManager 发送 Notification。

下面通过实例来介绍 Notification 的功能和用法。

实例：加薪通知

本实例示范了如何通过 NotificationManager 来发送、取消 Notification，本实例的界面很简单，只是包含两个普通按钮，分别用于发送 Notification 和取消 Notification。本示例程序的 Java 代码如下。

程序清单：codes\02\2.8\NotificationTest\src\org\crazyit\ui\NotificationTest.java

```
public class NotificationTest extends Activity
{
    static final int NOTIFICATION_ID = 0x123;
    NotificationManager nm;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 NotificationManager 服务
        nm = (NotificationManager)
            getSystemService(NOTIFICATION_SERVICE);
    }
    // 为发送通知的按钮的点击事件定义事件处理方法
    public void send(View source)
    {
        // 创建一个启动其他 Activity 的 Intent
        Intent intent = new Intent(NotificationTest.this
            , OtherActivity.class);
        PendingIntent pi = PendingIntent.getActivity(
            NotificationTest.this, 0, intent, 0);
        Notification notify = new Notification.Builder(this)
            // 设置打开该通知，该通知自动消失
            .setAutoCancel(true)
            // 设置显示在状态栏的通知提示信息
            .setTicker("有新消息")
```

```
                    // 设置通知的图标
                    .setSmallIcon(R.drawable.notify)
                    // 设置通知内容的标题
                    .setContentTitle("一条新通知")
                    // 设置通知内容
                    .setContentText("恭喜你，您加薪了，工资增加 20%!")
                    // 设置使用系统默认的声音、默认 LED 灯
                    // .setDefaults(Notification.DEFAULT_SOUND
                        // |Notification.DEFAULT_LIGHTS)
                    // 设置通知的自定义声音
                    .setSound(Uri.parse("android.resource://org.crazyit.ui/"
                        + R.raw.msg))
                    .setWhen(System.currentTimeMillis())
                    // 设改通知将要启动程序的 Intent
                    .setContentIntent(pi)
                    .build();
                // 发送通知
                nm.notify(NOTIFICATION_ID, notify);
        }
        // 为删除通知的按钮的点击事件定义事件处理方法
        public void del(View v)
        {
                // 取消通知
                nm.cancel(NOTIFICATION_ID);
        }
}
```

上面的程序中粗体字代码用于为 Notification 设置各种属性，包括 Notification 的图标、标题、发送时间等。

除此之外，上面的程序还通过 setDefaults()方法为 Notification 设置了声音提示、振动提示、闪光灯等。该属性支持如下属性值。

- DEFAULT_SOUND：设置使用默认声音。
- DEFAULT_VIBRATE：设置默认振动。
- DEFAULT_LIGHTS：设置使用默认闪光灯。
- ALL：设置使用默认声音、振动、闪光灯。

如果不想使用默认设置，也可以使用如下代码：

```
// 设置自定义声音
setSound(Uri.parse("file:///sdcard/click.mp3"));
//设置自定义振动
setVibrate(new long[]{0 , 50 , 100, 150});
```

接下来①号代码用于为该 Notification 设置事件信息，设置事件信息时传入了一个 PendingIntent 对象，该对象里封装了一个 Intent，这意味着单击该 Notification 时将会启动该 Intent 对应的程序。

运行上面的程序，单击程序中“发送 Notification”按钮，将可以看到手机屏幕上方出现了一个 Notification。将状态栏向下拖动，将可以看到 Notification 的详情，如图 2.68 所示。

图 2.68 所示的 Notification 还关联了一个 Activity：OtherActivity，因此当用户单击“普通通知”时即可启动 OtherActivity——OtherActivity 是一个十分简单的程序，故此处不再介绍。



图 2.68 发送通知

由于上面的程序指定了该 Notification 要启动 OtherActivity，因此一定不要忘记在 AndroidManifest.xml 文件中声明该 Activity。而且上面的程序还需要访问系统闪光灯、振动器，这也需要在 AndroidManifest.xml 文件中声明权限。也就是增加如下代码片

段即可：

```
<activity android:name=".OtherActivity" android:label="@string/other_activity">
</activity>
<!-- 添加操作闪光灯的权限 -->
<uses-permission android:name="android.permission.FLASHLIGHT"/>
<!-- 添加操作振动器的权限 -->
<uses-permission android:name="android.permission.VIBRATE"/>
```

2.9 对话框

Android 提供了丰富的对话框支持，它提供了如下 4 种常用的对话框。

- AlertDialog：功能最丰富、实际应用最广的对话框。
- ProgressDialog：进度对话框，这个对话框只是对简单进度条的封装。
- DatePickerDialog：日期选择对话框，这个对话框只是对 DatePicker 的包装。
- TimePickerDialog：时间选择对话框，这个对话框只是对 TimePicker 的包装。

上面 4 种对话框中功能最强、用法最灵活的就是 AlertDialog，因此应用也非常广泛，而其他三种对话框都是它的子类。图 2.69 显示了它们的继承关系类图。



图 2.69　AlertDialog 及其子类的继承关系类图

本节将会详细介绍各种对话框的功能和用法。

2.9.1 使用 AlertDialog 创建对话框

AlertDialog 的功能很强大，它可以生成各种内容的对话框。但实际上 AlertDialog 生成的对话框总有如图 2.70 所示结构。

从图 2.70 可以看出，AlertDialog 生成的对话框可分为如下 4 个区域。



图 2.70　对话框的结构

- 图标区。
- 标题区。
- 内容区。
- 按钮区。

从上面对话框的结构来看，创建一个对话框需要经过如下几步。

1. 使用创建 AlertDialog.Builder 对象。
2. 调用 AlertDialog.Builder 的 setTitle()或 setCustomTitle()方法设置标题。
3. 调用 AlertDialog.Builder 的 setIcon()方法设置标题。
4. 调用 AlertDialog.Builder 的相关设置方法设置对话框内容。
5. 调用 AlertDialog.Builder 的 setPositiveButton()、setNegativeButton()或 setNeutralButton()方法添加多个按钮。

6 调用 AlertDialog.Builder 的 create()方法创建 AlertDialog 对象，再调用 AlertDialog 对象的 show()方法将该对话框显示出来。

在上面的 6 个步骤中，第 4 个步骤是最灵活的，AlertDialog 允许创建各种内容的对话框。归纳起来，AlertDialog 提供了如下 6 种方法来指定对话框的内容。

- setMessage()：设置对话内容为简单文本内容。
- setItems()：设置对话框内容为简单列表项。
- setSingleChoiceItems()：设置对话框内容为单选列表项。
- setMultiChoiceItems()：设置对话框内容为多选列表项。
- setAdapter()：设置对话框内容为自定义列表项。
- setView()：设置对话框内容为自定义 View。

下面通过几个实例来介绍 AlertDialog 的用法。

实例：显示提示消息的对话框

本程序的界面非常简单，程序界面上定义了 1 个文本框和 6 个按钮，每次用户单击一个按钮时，将会显示不同类型的对话框。

本实例的界面只包含一个简单的文本框和几个按钮。当用户单击按钮时将会显示对话框。由于界面十分简单，故此处不再给出界面布局文件。

本程序将会通过上面介绍的 4 步来创建 AlertDialog 对话框。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
public void simple(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("简单对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        .setMessage("对话框的测试内容\n 第二行内容");
    // 为 AlertDialog.Builder 添加“确定”按钮
    setPositiveButton(builder);
    // 为 AlertDialog.Builder 添加“取消”按钮
    setNegativeButton(builder)
        .create()
        .show();
}
```

上面的程序中粗体字代码为该对话框设置了图标、标题等属性，上面的程序中还调用了 setPositiveButton()和 setNegativeButton()方法添加按钮。这两个方法的代码如下。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
private AlertDialog.Builder setPositiveButton(
        AlertDialog.Builder builder)
{
    // 调用 setPositiveButton 方法添加“确定”按钮
    return builder.setPositiveButton("确定", new OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog, int which)
        {
            show.setText("单击了【确定】按钮！");
        }
```

```
        });
    }
    private AlertDialog.Builder setNegativeButton(
            AlertDialog.Builder builder)
    {
        // 调用 setNegativeButton 方法添加“取消”按钮
        return builder.setNegativeButton("取消", new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                show.setText("单击了【取消】按钮！");
            }
        });
    }
```



图 2.71　简单对话框

除此之外，AlertDialog.Builder 还提供了如下方法来添加按钮。

setNeutralButton(CharSequence text, DialogInterface. OnClick Listener listener)：添加一个装饰性按钮。

因此 Android 的对话框一共可以生成三个对话框。

运行上面的程序，单击程序中“简单对话框”按钮，将看到如图 2.71 所示界面。

实例：简单列表项对话框

AlertDialog.Builder 调用 setItems()方法即可设置简单列表项对话框，调用该方法时需要传入一个数组或数组资源的资源 ID。

下面的代码调用 setItems()来设置简单列表项对话框，程序代码如下。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
public void simpleList(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("简单列表对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置简单的列表项内容
        .setItems(items, new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                show.setText("你选中了《" + items[which] + "》");
            }
        });
    // 为 AlertDialog.Builder 添加“确定”按钮
    setPositiveButton(builder);
    // 为 AlertDialog.Builder 添加“取消”按钮
    setNegativeButton(builder)
        .create()
        .show();
}
```

上面的程序中粗体字代码调用 AlertDialog.Builder 的 setItems()方法为对话框设置了多个列表项，此处生成了 4 个普通列表项。

运行上面的程序后单击“简单列表项对话框”按钮，程序将显示如图 2.72 所示的界面。



图 2.72 简单列表项对话框

实例：单选列表项对话框

只要调用 AlertDialog.Builder 的 setSingleChoiceItems()方法即可创建带单选列表项的对话框。调用 setSingleChoiceItems()方法时既可传入数组作为参数，也可传入 Cursor（相当于数据库查询结果集）作为参数，也可传入 ListAdapter 作为参数。如果传入 ListAdapter 作为参数，则由 ListAdapter 来提供多个列表项组件。

下面的方法调用了 setSingleChoiceItems()方法来创建带单选列表项的对话框。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
public void singleChoice(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("单选列表项对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置单选列表项，默认选中第二项（索引为 1）
        .setSingleChoiceItems(items, 1, new OnClickListener()
        {
            @Override
            public void onClick(DialogInterface dialog, int which)
            {
                show.setText("你选中了《" + items[which] + "》");
            }
        });
    // 为 AlertDialog.Builder 添加“确定”按钮
    setPositiveButton(builder);
    // 为 AlertDialog.Builder 添加“取消”按钮
    setNegativeButton(builder)
        .create()
        .show();
}
```

运行上面的程序，单击“单选列表项对话框”按钮，应用显示如图 2.73 所示的界面。



图 2.73 单选列表项对话框

实例：多选列表项对话框

只要调用 AlertDialog.Builder 的 setMultieChoiceItems()方法即可创建一个多选列表的对话框。调用 setMultieChoiceItems()方法时既可传入数组作为参数，也可传入 Cursor（相当于数据库查询结果集）作为参数。

下面的方法调用了 setMultieChoiceItems()方法来创建带多选列表项的对话框。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
public void multiChoice(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("多选列表项对话框")
        // 设置图标
```

```
            .setIcon(R.drawable.tools)
            // 设置多选列表项，设置勾选第 2 项、第 4 项
            .setMultiChoiceItems(items
                  , new boolean[]{false , true ,false ,true}, null);
        // 为 AlertDialog.Builder 添加“确定”按钮
        setPositiveButton(builder);
        // 为 AlertDialog.Builder 添加“取消”按钮
        setNegativeButton(builder)
            .create()
                                    .show();
    }
```



图 2.74　多选列表项对话框

调用 AlertDialog.Builder 的 setMultiChoiceItems()方法添加多选列表时，需要传入一个 boolean[]参数，该参数有两个作用：① 设置初始化时选中哪些列表项。② 该 boolean[]类型的参数还可用于动态地获取多选列表中列表项的选中状态。

运行上面的程序，然后单击“多选列表项对话框”按钮，程序将显示如图 2.74 所示的界面。

实例：自定义列表项对话框

AlertDialog.Builder 提供了一个 setAdapter()方法来设置对话框的内容，该方法需要传入一个 Adapter 参数，这样即可由该 Adapter 负责提供多个列表项组件。



提示：

不仅 setAdapter()方法可以接受 Adapter 作为参数，setSingleChoice()方法也可以接受 Adapter 作为参数，这意味着调用 setSingleChoice()方法也可实现自定义列表项对话框。

下面的方法调用了 setAdapter()方法来创建自定义列表项的对话框。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
public void customList(View source)
{
    AlertDialog.Builder builder = new AlertDialog.Builder(this)
        // 设置对话框标题
        .setTitle("自定义列表项对话框")
        // 设置图标
        .setIcon(R.drawable.tools)
        // 设置自定义列表项
        .setAdapter(new ArrayAdapter<String>(this
                , R.layout.array_item
                , items), null);
    // 为 AlertDialog.Builder 添加“确定”按钮
    setPositiveButton(builder);
    // 为 AlertDialog.Builder 添加“取消”按钮
    setNegativeButton(builder)
        .create()
        .show();
}
```

上面的程序中粗体字代码调用 setAdapter()方法时传入了一个 ArrayAdapter，该 ArrayAdapter 将会负责提供多个自定义列表项。如果需要，完全可以用创建 SimpleAdapter 对象或扩展 BaseAdapter 的方式来实现 Adapter，并作为参数传入 setAdapter()方法。

运行上面的程序，然后单击“自定义列表项对话框”按钮，程序将显示如图 2.75 所示的界面。



图 2.75 自定义列表项对话框

实例：自定义 View 的对话框

AlertDialog.Builder 的 setView()方法可以接受一个 View 组件，该 View 组件将会作为对话框的内容，通过这种方式，开发者可以“随心所欲”地定制对话框的内容——因为在 Android 界面编程中，一切都是 View。

本实例将会定义一个登录对话框，为实现该登录对话框，先定义一个登录的界面布局，该界面的布局文件如下。

程序清单：codes\02\2.9\AlertDialogTest\res\layout\login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/loginForm"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<TableRow>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="用户名:"
      android:textSize="10pt"
      />
<!-- 输入用户名的文本框 -->
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="请填写登录账号"
      android:selectAllOnFocus="true"
      />
</TableRow>
<TableRow>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="密码:"
      android:textSize="10pt"
      />
<!-- 输入密码的文本框 -->
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="请填写密码"
      android:password="true"
      />
</TableRow>
<TableRow>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="电话号码:"
      android:textSize="10pt"
      />
```

```
<!-- 输入电话号码的文本框 -->
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="请填写您的电话号码"
      android:selectAllOnFocus="true"
      android:phoneNumber="true"
      />
</TableRow>
</TableLayout>
```

上面的界面布局定义了登录用的三个输入框：输入用户名的文本框、输入密码的密码框、输入电话号码的输入框。接下来在应用程序中调用 AlertDialog.Builder 的 setView(View view)方法让对话框显示该输入界面即可。

该程序与前面介绍的列表对话框程序比较相似，只是将原来的调用 setItems()设置列表项，改为现在的调用 setView()来设置自定义视图。下面给出该程序的关键代码。

程序清单：codes\02\2.9\AlertDialogTest\src\org\crazyit\ui\AlertDialogTest.java

```
public void customView(View source)
{
      //装载/res/layout/login.xml 界面布局
      TableLayout loginForm = (TableLayout)getLayoutInflater()
            .inflate( R.layout.login, null);
      new AlertDialog.Builder(this)
            // 设置对话框的图标
            .setIcon(R.drawable.tools)
            // 设置对话框的标题
            .setTitle("自定义 View 对话框")
            // 设置对话框显示的 View 对象
            .setView(loginForm)
            // 为对话框设置一个“确定”按钮
            .setPositiveButton("登录" , new OnClickListener()
            {
                  @Override
                  public void onClick(DialogInterface dialog,
                              int which)
                  {
                        // 此处可执行登录处理
                  }
            })
            // 为对话框设置一个“取消”按钮
            .setNegativeButton("取消", new OnClickListener()
            {
                  @Override
                  public void onClick(DialogInterface dialog,
                              int which)
                  {
                        // 取消登录，不做任何事情
                  }
            })
            // 创建并显示对话框
            .create()
            .show();
}
```

上面的程序中第一行粗体字代码显式装载了/res/layout/ login.xml 文件，并返回该文件对应的 TableLayout 作为 View，接下来程序调用了 AlertDialog.Builder 的 setView()方法来显示上一行代码所获得的 TableLayout。

运行上面的程序，然后单击“自定义 View 对话框”按钮，应用程序将显示如图 2.76 所示的界面。



图 2.76 自定义 View 对话框

2.9.2 对话框风格的窗口

还有一种自定义对话框的方式，这种对话框本质上依然是窗口，只是把显示窗口的 Activity 的风格设为对话框风格即可。

下面的程序定义一个简单的界面布局，该界面布局里包含一个 ImageView 和一个 Button。接下来程序使用 Activity 来显示该界面布局。

程序清单：codes\02\2.9\DialogTheme\src\org\crazyit\ui\DialogTheme.java

```
public class DialogTheme extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 结束该 Activity
                finish();
            }
        });
    }
}
```

上面的程序仅仅是为界面上的按钮绑定了一个监听器，当该按钮被单击时结束该 Activity。

接下来在 AndroidManifest.xml 文件中指定该窗口以对话框风格显示，也就是在该清单文件中使用如下配置片段：

```
<activity android:name=".DialogTheme "
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Dialog">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

上面的粗体字代码指定 DialogTheme 使用对话框风格进行显示。运行上面的程序，将看到如图 2.77 所示界面。



图 2.77 对话框风格的 Activity

2.9.3 使用 PopupWindow

PopupWindow 可以创建类似对话框风格的窗口，使用 PopupWindow 创建对话框风格的窗口只要如下两步即可：

① 调用 PopupWindow 的构造器创建 PopupWindow 对象。

② 调用 PopupWindow 的 showAsDropDown(View v)将 PopupWindow 作为 v 组件的下拉组件显示出来；或调用 PopupWindow 的 showAtLocation()方法将 PopupWindow 在指定位置显示出来。

下面的程序示范了如何使用 PopupWindow 创建对话框风格的窗口。该程序的主程序中只有一个简单的按钮，用户单击该按钮时将会显示 PopupWindow；其中 PopupWindow 负责加载并显示前一个示例的窗口界面。

该程序代码如下。

程序清单：codes\02\2.9\PopupWindowTest\src\org\crazyit\ui\PopupWindowTest.java

```
public class PopupWindowTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 装载 R.layout.popup 对应的界面布局
        View root = this.getLayoutInflater().inflate(R.layout.popup, null);
        // 创建 PopupWindow 对象
        final PopupWindow popup = new PopupWindow(root, 280, 360);
        Button button = (Button) findViewById(R.id.bn);
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 以下拉方式显示
                // popup.showAsDropDown(v);
                //将 PopupWindow 显示在指定位置
                popup.showAtLocation(findViewById(R.id.bn), Gravity.CENTER, 20,
                    20);
            }
        });
        // 获取 PopupWindow 中的“关闭”按钮
        root.findViewById(R.id.close).setOnClickListener(
            new View.OnClickListener()
            {
                public void onClick(View v)
                {
                    // 关闭 PopupWindow
                    popup.dismiss(); //①
                }
            });
    }
}
```



图 2.78　PopupWindow 效果

上面的程序中第一行粗体字代码用于创建 PopupWindow 对象，接下来的两行粗体字代码分别示范了两种显示 PopupWindow 的方式：以下拉方式显示和在指定位置显示，这就是在程序中创建并显示 PopupWindow 的关键代码。程序中①号粗体字代码负责销毁、隐藏 PopupWindow 对象——当用户单击 PopupWindow 中的“关闭”按钮时，该 PopupWindow 将会关闭。

运行上面的程序，将可以看到如图 2.78 所示结果。

从图 2.78 所示效果不难看出，PopupWindow 非常适合显示一些需要浮动显示的内容。

2.9.4 使用 DatePickerDialog、TimePickerDialog

DatePickerDialog 与 TimePickerDialog 的功能比较简单，用法也简单，只要如下两步即可。

① 通过 new 关键字创建 DatePickerDialog、TimePickerDialog 实例，调用它们的 show() 方法即可将日期选择对话框、时间选择对话框显示出来。

② 为 DatePickerDialog、TimePickerDialog 绑定监听器，这样可以保证用户通过 DatePickerDialog、TimePickerDialog 设置事件时触发监听器，从而通过监听器来获取用户设置的事件。

下面程序的界面上定义了两个按钮，一个按钮用于打开日期选择对话框，一个用于打开时间选择对话框。该程序的界面定义文件非常简单，故不再给出。该程序的 Java 代码如下。

程序清单：codes\02\2.9\DateDialogTest\src\org\crazyit\ui\DateDialogTest.java

```
public class DateDialogTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button dateBn = (Button)findViewById(R.id.dateBn);
        Button timeBn = (Button)findViewById(R.id.timeBn);
        //为“设置日期”按钮绑定监听器
        dateBn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Calendar c = Calendar.getInstance();
                // 直接创建一个 DatePickerDialog 对话框实例，并将它显示出来
                new DatePickerDialog(DateDialogTest.this,
                    // 绑定监听器
                    new DatePickerDialog.OnDateSetListener()
                    {
                        @Override
                        public void onDateSet(DatePicker dp, int year,
                            int month, int dayOfMonth)
                        {
                            EditText show = (EditText) findViewById
                            (R.id.show);
                            show.setText("您选择了：" + year + "年" + (month
                            + 1)
                                + "月" + dayOfMonth + "日");
                        }
                    }
                //设置初始日期
                , c.get(Calendar.YEAR)
                , c.get(Calendar.MONTH)
                , c.get(Calendar.DAY_OF_MONTH)).show();
            }
        });
        //为“设置时间”按钮绑定监听器
        timeBn.setOnClickListener(new OnClickListener()
        {
            @Override
```

```
                public void onClick(View source)
                {
                    Calendar c = Calendar.getInstance();
                    // 创建一个 TimePickerDialog 实例，并把它显示出来
                    new TimePickerDialog(DateDialogTest.this,
                        // 绑定监听器
                        new TimePickerDialog.OnTimeSetListener()
                        {
                            @Override
                            public void onTimeSet(TimePicker tp, int hourOfDay,
                                int minute)
                            {
                                EditText show = (EditText) findViewById(R.id.
                                show);
                                show.setText("您选择了：" + hourOfDay + "时" +
                                minute
                                    + "分");
                            }
                        }
                    //设置初始时间
                    , c.get(Calendar.HOUR_OF_DAY)
                    , c.get(Calendar.MINUTE)
                    //true 表示采用 24 小时制
                    , true).show();
                }
            });
        }
    }
```



图 2.79　日期选择对话框

上面的程序中两段粗体字代码就是创建并显示 DatePickerDialog、TimePickerDialog 的关键代码。运行上面的程序，如果单击“设置日期”按钮，系统将会显示如图 2.79 所示的日期选择对话框。

如果用户单击程序中的“设置时间”对话框，系统将会显示日期选择对话框。需要指出的是，日期选择对话框、时间选择对话框只是供用户来选择日期、时间，对 Android 的系统日期、时间没有任何影响。



提示：

Android 暂时还没有公开设置系统日期、时间的 API，如果需要在程序中设置 Android 系统日期、时间，目前所知的方式都需要重新编译 Android 系统源代码，比较烦琐。

2.9.5　使用 ProgressDialog 创建进度对话框

ProgressDialog 代表了进度对话框，程序只要创建 ProgressDialog 实例，并将它显示出来就是一个进度对话框。使用 ProgressDialog 创建进度对话框有如下两种方式。

① 如果只是创建简单的进度对话框，调用 ProgressDialog 提供的静态 show()方法显示对话框即可。

② 创建 ProgressDialog，然后调用方法对对话框里的进度条进行设置，设置完成后将对话框显示出来即可。

为了对进度对话框里的进度条进行设置，ProgressDialog 包含了如下常用的方法。

- setIndeterminate(boolean indeterminate)：设置对话框里的进度条不显示进度值。
- setMax(int max)：设置对话框里进度条的最大值。
- setMessage(CharSequence message)：设置对话框里显示的消息。
- setProgress(int value)：设置对话框里进度条的进度值。
- setProgressStyle(int style)：设置对话框里进度条的风格。

下面的程序的界面也很简单，界面上只有三个简单的按钮，当用户单击不同按钮时系统将会启动不同的进度对话框。其中第三个按钮激发的进度对话框比较复杂，该对话框使用填充数组来模拟耗时任务，随着任务进行不断更新进度对话框上进度的显示。

程序清单：codes\02\2.9\ProgressDialogTest\src\org\crazyit\ui\ProgressDialogTest.java

```
public class ProgressDialogTest extends Activity
{
    final static int MAX_PROGRESS = 100;
    // 该程序模拟填充长度为 100 的数组
    private int[] data = new int[50];
    // 记录进度对话框的完成百分比
    int progressStatus = 0;
    int hasData = 0;
    ProgressDialog pd1,pd2;
    // 定义一个负责更新的进度的 Handler
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            // 表明消息是由该程序发送的
            if (msg.what == 0x123)
            {
                pd2.setProgress(progressStatus);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void showSpinner(View source)
    {
        // 调用静态方法显示环形进度条
        ProgressDialog.show(this, "任务执行中"
            , "任务执行中，请等待", false, true); //①
    }
    public void showIndeterminate(View source)
    {
        pd1 = new ProgressDialog(ProgressDialogTest.this);
        // 设置对话框的标题
        pd1.setTitle("任务正在执行中");
        // 设置对话框显示的内容
        pd1.setMessage("任务正在执行中，敬请等待...");
        // 设置对话框能用“取消”按钮关闭
        pd1.setCancelable(true);
        // 设置对话框的进度条风格
        pd1.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        // 设置对话框的进度条是否显示进度
```

```
        pd1.setIndeterminate(true);
        pd1.show(); //②
    }
    public void showProgress(View source)
    {
        // 将进度条的完成进度重设为 0
        progressStatus = 0;
        // 重新开始填充数组
        hasData = 0;
        pd2 = new ProgressDialog(ProgressDialogTest.this);
        pd2.setMax(MAX_PROGRESS);
        // 设置对话框的标题
        pd2.setTitle("任务完成百分比");
        // 设置对话框显示的内容
        pd2.setMessage("耗时任务的完成百分比");
        // 设置对话框不能用“取消”按钮关闭
        pd2.setCancelable(false);
        // 设置对话框的进度条风格
        pd2.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
        // 设置对话框的进度条是否显示进度
        pd2.setIndeterminate(false);
        pd2.show(); //③
        new Thread()
        {
            public void run()
            {
                while (progressStatus < MAX_PROGRESS)
                {
                    // 获取耗时操作的完成百分比
                    progressStatus = MAX_PROGRESS
                        * doWork() / data.length;
                    // 发送空消息到 Handler
                    handler.sendEmptyMessage(0x123);
                }
                // 如果任务已经完成
                if (progressStatus >= MAX_PROGRESS)
                {
                    // 关闭对话框
                    pd2.dismiss();
                }
            }
        }.start();
    }
    // 模拟一个耗时的操作
    public int doWork()
    {
        // 为数组元素赋值
        data[hasData++] = (int) (Math.random() * 100);
        try
        {
            Thread.sleep(100);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }
        return hasData;
    }
}
```

上面程序为三个按钮定义了事件处理方法，其中一个按钮的事件处理方法调用了 ProgressDialog 的静态 show()方法创建并显示对话框，如上面①号粗体字代码所示，这也是使

用对话框的最简单的方式；第二个按钮的事件处理方法先创建了 ProgressDialog 对象，再调用该对话框的 show()方法将它显示出来，如上面②号粗体字代码所示；第三个按钮的事件处理方法也是先创建 ProgressDialog 对象，并设置其中对话框的相关属性，再调用 show()方法将它显示出来，如上面③号代码所示。

单击上面的程序中第一个按钮，将看到如图 2.80 所示界面。

单击第三个按钮，将可以看到如图 2.81 所示界面。



图 2.80 环形风格的进度条



图 2.81 水平风格的进度条

2.10 菜单

菜单在桌面应用中使用十分广泛，几乎所有的桌面应用都有菜单。菜单在手机应用中的使用减少了不少（主要受到手机屏幕大小制约），但依然有不少手机应用会添加菜单。

与桌面应用的菜单不同，Android 应用中的菜单默认是看不见的，只有当用户单击手机上的“MENU”键（位于模拟器右边的物理键盘上）时，系统才会显示该应用关联的菜单，这种菜单叫选项菜单（Option Menu）。

注意：

从 Android 3.0 开始，Android 并不要求手机设备上必须提供 MENU 按键，可能部分 Android 手机将不再提供 MENU 按键。在这样的情况下，Android 推荐使用 ActionBar 来代替菜单。下一节会介绍 Android 的 ActionBar 支持。



Android 应用同样支持上下文菜单（ContextMenu），当用户一直按住某个组件时，该组件所关联的上下文菜单就显示出来。

2.10.1 选项菜单和子菜单（SubMenu）

为了让读者感受 Android 应用中菜单的外观和功能，先简单看一下 Android 系统自带的选项菜单，按如下两步进行。

① 单击模拟器右边的按键（返回桌面），系统返回桌面。

② 单击模拟器右边的“MENU”按键，将可以在手机屏幕下方看到系统默认的选项菜单，如图 2.82 所示。

从图 2.82 可以看出，Android 的选项菜单默认是看不见的，当用户按下“MENU”键时程序菜单将会出现在屏幕下方。

Android 系统的菜单支持主要通过 4 个接口来体现，图 2.83 显示了 Android 菜单支持的 4 个接口。



图 2.82　选项菜单

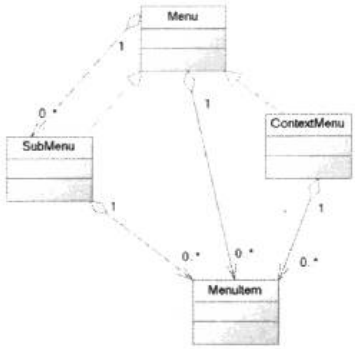


图 2.83　Android 系统的菜单支持

从图 2.83 可以看出，Menu 接口只是一个父接口，该接口下有如下两个子接口。

- SubMenu：它代表一个子菜单。可以包含 1～*N* 个 MenuItem（形成菜单项）。
- ContextMenu：它代表一个上下文菜单。可以包含 1～*N* 个 MenuItem（形成菜单项）。

Android 的不同菜单具有如下特征。

- 选项菜单：选项菜单不支持勾选标记，并且只显示菜单的“浓缩（condensed）”标题。
- 子菜单（SubMenu）：不支持菜单项图标，不支持嵌套子菜单。
- 上下文菜单（ContextMenu）：不支持菜单快捷键和图标。

Menu 接口定义了如下方法来添加菜单或菜单项。

- MenuItem add(int titleRes)：添加一个新的菜单项。
- MenuItem add(int groupId, int itemId, int order, CharSequence title)：添加一个新的处于 groupId 组的菜单项。
- MenuItem add(int groupId, int itemId, int order, int titleRes)：添加一个新的处于 groupId 组的菜单项。
- MenuItem add(CharSequence title)：添加一个新的菜单项。
- SubMenu addSubMenu(int titleRes)：添加一个新的子菜单。
- SubMenu addSubMenu(int groupId, int itemId, int order, int titleRes)：添加一个新的处于 groupId 组的子菜单。
- SubMenu addSubMenu(CharSequence title)：添加一个新的子菜单。
- SubMenu addSubMenu(int groupId, int itemId, int order, CharSequence title)：添加一个新的处于 groupId 组的子菜单。

上面的方法归纳起来就是两个：add()方法用于添加菜单项；addSubMenu()用于添加子菜单。这些重载方法的区别只是：是否将子菜单、菜单项添加到指定菜单组中，是否使用资源文件中的字符串资源来设置标题。

SubMenu 继承了 Menu，它就代表了一个子菜单，额外提供了如下常用方法。

- SubMenu setHeaderIcon(Drawable icon)：设置菜单头的图标。
- SubMenu setHeaderIcon(int iconRes)：设置菜单头的图标。
- SubMenu setHeaderTitle(int titleRes)：设置菜单头的标题。
- SubMenu setHeaderTitle(CharSequence title)：设置菜单头的标题。
- SubMenu setHeaderView(View view)：使用 View 来设置菜单头。

掌握了上面 Menu、SubMenu、MenuItem 的用法之后，接下来就可通过它们来为 Android

应用添加菜单或子菜单了。添加菜单或子菜单的步骤如下。

① 重写 Activity 的 onCreateOptionsMenu(Menu menu)的方法，在该方法里调用 Menu 对象的方法来添加菜单项或子菜单。

② 如果希望应用程序能响应菜单项的单击事件，重写 Activity 的 onOptionsItemSelected (MenuItem mi)方法即可。

下面的程序示范了如何为 Android 应用添加菜单和子菜单。该程序的界面布局很简单，故不给出界面布局的文件。该程序的 Java 代码如下。

程序清单：codes\02\2.10\MenuTest\src\org\crazyit\ui\MenuTest.java

```
public class MenuTest extends Activity
{
    // 定义字体大小菜单项的标识
    final int FONT_10 = 0x111;
    final int FONT_12 = 0x112;
    final int FONT_14 = 0x113;
    final int FONT_16 = 0x114;
    final int FONT_18 = 0x115;
    // 定义普通菜单项的标识
    final int PLAIN_ITEM = 0x11b;
    // 定义字体颜色菜单项的标识
    final int FONT_RED = 0x116;
    final int FONT_BLUE = 0x117;
    final int FONT_GREEN = 0x118;
    private EditText edit;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        edit = (EditText) findViewById(R.id.txt);
    }
    // 当用户单击 MENU 键时触发该方法
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // -------------向 menu 中添加字体大小的子菜单-------------
        SubMenu fontMenu = menu.addSubMenu("字体大小");
        // 设置菜单的图标
        fontMenu.setIcon(R.drawable.font);
        // 设置菜单头的图标
        fontMenu.setHeaderIcon(R.drawable.font);
        // 设置菜单头的标题
        fontMenu.setHeaderTitle("选择字体大小");
        fontMenu.add(0, FONT_10, 0, "10 号字体");
        fontMenu.add(0, FONT_12, 0, "12 号字体");
        fontMenu.add(0, FONT_14, 0, "14 号字体");
        fontMenu.add(0, FONT_16, 0, "16 号字体");
        fontMenu.add(0, FONT_18, 0, "18 号字体");
        // -------------向 menu 中添加普通菜单项-------------
        menu.add(0, PLAIN_ITEM, 0, "普通菜单项");
        // -------------向 menu 中添加文字颜色的子菜单-------------
        SubMenu colorMenu = menu.addSubMenu("字体颜色");
        colorMenu.setIcon(R.drawable.color);
        // 设置菜单头的图标
        colorMenu.setHeaderIcon(R.drawable.color);
        // 设置菜单头的标题
        colorMenu.setHeaderTitle("选择文字颜色");
```

```
        colorMenu.add(0, FONT_RED, 0, "红色");
        colorMenu.add(0, FONT_GREEN, 0, "绿色");
        colorMenu.add(0, FONT_BLUE, 0, "蓝色");
        return super.onCreateOptionsMenu(menu);
    }
    @Override
    // 选项菜单的菜单项被单击后的回调方法
    public boolean onOptionsItemSelected(MenuItem mi)
    {
        //判断单击的是哪个菜单项，并有针对性地作出响应
        switch (mi.getItemId())
        {
            case FONT_10:
                edit.setTextSize(10 * 2);
                break;
            case FONT_12:
                edit.setTextSize(12 * 2);
                break;
            case FONT_14:
                edit.setTextSize(14 * 2);
                break;
            case FONT_16:
                edit.setTextSize(16 * 2);
                break;
            case FONT_18:
                edit.setTextSize(18 * 2);
                break;
            case FONT_RED:
                edit.setTextColor(Color.RED);
                break;
            case FONT_GREEN:
                edit.setTextColor(Color.GREEN);
                break;
            case FONT_BLUE:
                edit.setTextColor(Color.BLUE);
                break;
            case PLAIN_ITEM:
                Toast toast = Toast.makeText(MenuTest.this
                    , "您单击了普通菜单项" , Toast.LENGTH_SHORT);
                toast.show();
                break;
        }
        return true;
    }
}
```

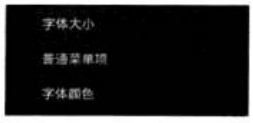


图 2.84　选项菜单

上面的程序中粗体字代码就是添加三个菜单的代码，三个菜单中有两个是子菜单，而且程序还为子菜单设置了图标、标题等。运行上面的程序，单击“MENU”按键，将看到程序下方显示如图 2.84 所示的菜单。

图 2.84 所示菜单中字体大小包含子菜单，普通菜单项只是一个菜单项，字体颜色也包含子菜单，如果开发者单击“字体颜色”菜单，将看到屏幕上显示如图 2.85 所示的子菜单。

由于程序重写了 onOptionsItemSelected(MenuItem mi)方法，因此当用户单击指定菜单项时，程序可以为菜单项的单击事件提供响应。

前面已经介绍过，如果程序要监听菜单项的单击事件，可以通过重写 onOptionsItemSelected(MenuItem item)方法来实现——每当用户单击任意菜单时，该方法都会

被触发。如果开发者需要针对不同菜单提供响应，就需要在 onOptionsItemSelected(MenuItem item)方法中进行判断了，如上面的程序使用 switch 语句进行了判断。由于程序需要在 onOptionsItemSelected(MenuItem item)方法中准确判断到底是哪个菜单项被单击了，因此添加菜单项时通常应为每个菜单项指定 ID。



图 2.85 子菜单

2.10.2 使用监听器来监听菜单事件

除了重写 onOptionsItemSelected(MenuItem item)方法来为菜单单击事件编写响应之外，Android 同样允许开发者为不同菜单分别绑定监听器。为菜单项绑定监听器的方法为：

setOnMenuItemClickListener(MenuItem.OnMenuItemClickListener menuItemClickListener)

在这种方式下，我们可以采用简单方法来添加菜单项，无须为每个菜单项目指定 ID。

一般来说，通过重写 onOptionsItemSelected(MenuItem mi)方法来使处理菜单的单击事件更加简洁，因为所有的事件处理代码都控制在该方法内，只需要判断到底单击了哪个菜单项。通过为每个菜单绑定事件监听器使得代码更加臃肿。因此，一般并不推荐为每个菜单项分别绑定监听器。

2.10.3 创建复选菜单项和单选菜单项

如果希望所创建的菜单项是单选菜单项或多选菜单项，则可以调用 MenuItem 的如下方法。

- setCheckable(boolean checkable)：设置该菜单项是否可以被勾选。

调用上面的方法后的菜单项默认是多选菜单项。

如果希望将一组菜单里的菜单项都设为可勾选的菜单项，则可调用如下方法。

- setGroupCheckable(int group, boolean checkable, boolean exclusive)：设置 group 组里的所有菜单项是否可勾选；如果将 exclusive 设为 true，那么它们将是一组单选菜单项。

除此之外，Android 还为 MenuItem 提供了如下方法来设置快捷键。

- setAlphabeticShortcut(char alphaChar)：设置字母快捷键。
- setNumericShortcut(char numericChar)：设置数字快捷键。
- setShortcut(char numericChar, char alphaChar)：同时设置两种快捷键。

2.10.4 设置与菜单项关联的 Activity

有些时候，应用程序需要单击某个菜单项时启动其他 Activity（包括其他 Service）。对于这种需求，Android 甚至不需要开发者编写任何事件处理代码，只要调用 MenuItem 的 setIntent(Intent intent)方法即可——该方法把该菜单项与指定 Intent 关联到一起，当用户单击该菜单项时，该 Intent 所代表的组件将会被启动。

如下程序示范了如何通过菜单项来启动指定 Activity。该程序几乎不包含任何界面组件，因此不给出界面布局文件。该程序的 Java 文件如下。

程序清单：codes\02\2.10\ActivityMenu\src\org\crazyit\ui\ActivityMenu.java

```
public class ActivityMenu extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // -------------向 menu 中添加子菜单-------------
        SubMenu prog = menu.addSubMenu("启动程序");
        // 设置菜单头的图标
        prog.setHeaderIcon(R.drawable.tools);
        // 设置菜单头的标题
        prog.setHeaderTitle("选择您要启动的程序");
        // 添加菜单项
        MenuItem item = prog.add("查看经典 Java EE");
        //为菜单项设置关联的 Activity
        item.setIntent(new Intent(this , OtherActivity.class));
        return super.onCreateOptionsMenu(menu);
    }
}
```

上面的程序中粗体字代码指定单击指定菜单项时启动 OtherActivity，因此程序还应该定义 OtherActivity，并在 AndroidManifest.xml 文件中添加如下代码来声明 OtherActivity：

```
<activity android:name=".OtherActivity"
    android:label="查看经典 Java EE">
</activity>
```



图 2.86　启动 Activity 的菜单项

运行上面的程序，打开“启动程序”菜单，看到如图 2.86 所示的子菜单。

单击图 2.86 所示子菜单中的“查看经典 Java EE”即可启动另一个 Activity：OtherActivity。

2.10.5　上下文菜单

Android 用 ContextMenu 来代表上下文菜单，为 Android 应用开发上下文菜单与开发选项菜单的方法基本相似，因为 ContextMenu 继承了 Menu，因此程序可用相同的方法为它添加菜单项。

当然，开发上下文菜单与开发选项菜单的区别在于：开发上下文菜单不是重写 onCreateOptionsMenu(Menu menu)方法，而是重写 onCreateContextMenu (ContextMenu menu, View source, ContextMenu.ContextMenuInfo menuInfo)方法。其中 source 参数代表触发上下文菜单的组件。

开发上下文菜单的步骤如下。

① 重写 Activity 的 onCreateContextMenu(ContextMenu menu, View source, ContextMenu. Context MenuInfo menuInfo)方法。

② 调用 Activity 的 registerForContextMenu(View view)方法为 view 组件注册上下文菜单。

③ 如果希望应用程序能为菜单项提供响应，可以重写 onContextItemSelected (MenuItem mi)方法，或为指定菜单项绑定事件监听器。

ContextMenu 提供了如下方法，同样可以为上下文菜单设置图标、标题等。

- ContextMenu setHeaderIcon(Drawable icon)：为上下文菜单设置图标。
- ContextMenu setHeaderIcon(int iconRes)：为上下文菜单设置图标。
- ContextMenu setHeaderTitle(int titleRes)：为上下文菜单设置标题。

下面的程序示范了如何开发上下文菜单，该程序的用户界面同样很简单，故不再给出界面布局文件。下面是该程序的 Java 代码。

程序清单：codes\02\2.10\ContextMenuTest\src\org\crazyit\ui\ContextMenuTest.java

```
public class ContextMenuTest extends Activity
{
    // 为每个菜单定义一个标识
    final int MENU1 = 0x111;
    final int MENU2 = 0x112;
    final int MENU3 = 0x113;
    private TextView txt;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txt = (TextView) findViewById(R.id.txt);
        // 为文本框注册上下文菜单
        registerForContextMenu(txt); //①
    }
    // 创建上下文菜单时触发该方法
    @Override
    public void onCreateContextMenu(ContextMenu menu, View source,
        ContextMenu.ContextMenuInfo menuInfo)
    {
        menu.add(0, MENU1, 0, "红色");
        menu.add(0, MENU2, 0, "绿色");
        menu.add(0, MENU3, 0, "蓝色");
        // 将三个菜单项设为单选菜单项
        menu.setGroupCheckable(0, true, true);
        //设置上下文菜单的标题、图标
        menu.setHeaderIcon(R.drawable.tools);
        menu.setHeaderTitle("选择背景色");
    }
    // 上下文菜单的菜单项被单击时触发该方法
    @Override
    public boolean onContextItemSelected(MenuItem mi)
    {
        switch (mi.getItemId())
        {
            case MENU1:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.RED);
                break;
            case MENU2:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.GREEN);
                break;
            case MENU3:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.BLUE);
                break;
        }
        return true;
```

```
        }
    }
```

上面的程序重写了 onCreateContextMenu(ContextMenu menu, View source, ContextMenu. Context MenuInfo menuInfo)方法，该方法的内部为程序创建了一个上下文菜单。



图 2.87　上下文菜单

程序在①号代码处调用 registerForContextMenu(txt)为 txt 组件（一个文本框组件）注册了上下文菜单，这意味着当用户长按该组件时显示上下文菜单。上下文菜单如图 2.87 所示。

2.10.6　使用 XML 文件定义菜单

Android 提供了两种创建菜单的方式，一种是在 Java 代码中创建，一种是使用 XML 资源文件定义。上面的实例都是在 Java 代码中创建菜单，在 Java 代码中定义菜单存在如下不足。

- 在 Java 代码中定义菜单、菜单项，必然导致程序代码臃肿。
- 需要程序员采用硬编码方式为每个菜单项分配 ID、为每个菜单组分配 ID，这种方式导致应用可扩展性、可维护性降低。

一般推荐使用 XML 资源文件来定义菜单，这种方式可以提供更好的解耦。

菜单资源文件通常应该放在/res/menu 目录下，菜单资源的根元素通常是<menu.../>元素，<menu.../>元素无须指定任何属性。<menu.../>元素内可包含如下子元素。

- <item.../>元素：定义菜单项。
- <group.../>子元素：将多个<item.../>定义的菜单包装成一个菜单组。

<group.../>子元素用于控制整组菜单的行为，该元素可指定如下常用属性。

- checkableBehavior：指定该组菜单的选择行为。可指定为 none（不可选）、all（多选）和 single（单选）三个值。
- menuCategory：对菜单进行分类，指定菜单的优先级。有效值为 container、system、secondary 和 alternative。
- visible：指定该组菜单是否可见。
- enable：指定该组菜单是否可用。

<item.../>元素用于定义一份菜单项，<item.../>元素又可包含<menu.../>元素，位于<item.../>元素内部的<menu.../>就代表了子菜单。

<item.../>元素可指定如下常用属性。

- android:id：为菜单项指定一个唯一标识。
- android:title：指定菜单项的标题。
- android:icon：指定菜单项的图标。
- android:alphabeticShortcut：为菜单项指定字符快捷键。
- android:numericShortcut：为菜单项指定数字快捷键。
- android:checkable：设置该菜单项是否可选。
- android:checked：设置该菜单项是否已选中。
- android:visible：设置该菜单项是否可见。
- android:enable：设置该菜单项是否可用。

一旦在程序中定义了菜单资源后，接下来还是重写 onCreateOptionsMenu（用于创建选项

菜单）、onCreateContextMenu（用于创建上下文菜单）方法，在这些方法中调用 MenuInflater 对象的 inflate 方法装载指定资源对应的菜单即可。

接下来将会开发一个使用 XML 资源定义菜单的实例，本实例将会把前面开发的菜单示例程序改为使用 XML 资源定义菜单。

实例：使用 XML 资源定义菜单

本实例包含两种菜单：选项菜单和上下文菜单，其中选项菜单对应的 XML 资源文件如下。

程序清单：codes\02\2.10\MenuResTest\res\menu\my_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="@string/font_size"
        android:icon="@drawable/font">
        <menu>
            <!-- 定义一组单选菜单项 -->
            <group android:checkableBehavior="single">
                <!-- 定义多个菜单项 -->
                <item
                    android:id="@+id/font_10"
                    android:title="@string/font_10"/>
                <item
                    android:id="@+id/font_12"
                    android:title="@string/font_12"/>
                <item
                    android:id="@+id/font_14"
                    android:title="@string/font_14"/>
                <item
                    android:id="@+id/font_16"
                    android:title="@string/font_16"/>
                <item
                    android:id="@+id/font_18"
                    android:title="@string/font_18"/>
            </group>
        </menu>
    </item>
    <!-- 定义一个普通菜单项 -->
    <item android:id="@+id/plain_item"
        android:title="@string/plain_item">
    </item>
    <item android:title="@string/font_color"
        android:icon="@drawable/color">
        <menu>
            <!-- 定义一组普通的菜单项 -->
            <group>
                <!-- 定义三个菜单项 -->
                <item
                    android:id="@+id/red_font"
                    android:title="@string/red_title"/>
                <item
                    android:id="@+id/green_font"
                    android:title="@string/green_title"/>
                <item
                    android:id="@+id/blue_font"
                    android:title="@string/blue_title"/>
            </group>
        </menu>
```

```
        </item>
    </menu>
```

上面的菜单资源文件的<menu.../>元素里包含三个<item.../>子元素，这表明该菜单里包含三个菜单项。其中第一个、第三个都包含子菜单。

接下来再为该应用定义上下文菜单的资源文件，代码如下。

程序清单：codes\02\2.10\MenuResTest\res\menu\context.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 定义一组单选菜单项目 -->
    <group android:checkableBehavior="single">
        <!-- 定义三个菜单项 -->
        <item
            android:id="@+id/red"
            android:title="@string/red_title"
            android:alphabeticShortcut="r"/>
        <item
            android:id="@+id/green"
            android:title="@string/green_title"
            android:alphabeticShortcut="g"/>
        <item
            android:id="@+id/blue"
            android:title="@string/blue_title"
            android:alphabeticShortcut="b"/>
    </group>
</menu>
```

定义了上面两份菜单资源之后，接下来即可在 Activity 的 onCreateOptionsMenu、onCreate Context Menu 方法中加载这两份菜单资源。下面是程序中加载并显示两份菜单的 Java 代码。

程序清单：codes\02\2.10\MenuResTest\src\org\crazyit\ui\MenuResTest.java

```
public class MenuResTest extends Activity
{
    private TextView txt;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txt = (TextView) findViewById(R.id.txt);
        // 为文本框注册上下文菜单
        registerForContextMenu(txt);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater inflator = new MenuInflater(this);
        //装填 R.menu.my_menu 对应的菜单，并添加到 menu 中
        inflator.inflate(R.menu.my_menu, menu);
        return super.onCreateOptionsMenu(menu);
    }
    // 创建上下文菜单时触发该方法
    @Override
    public void onCreateContextMenu(ContextMenu menu, View source,
        ContextMenu.ContextMenuInfo menuInfo)
    {
        MenuInflater inflator = new MenuInflater(this);
        //装填 R.menu.context 对应的菜单，并添加到 menu 中
```

```
        inflator.inflate(R.menu.context , menu);
        menu.setHeaderIcon(R.drawable.tools);
        menu.setHeaderTitle("请选择背景色");
    }
    // 上下文菜单中菜单项被单击时触发该方法
    @Override
    public boolean onContextItemSelected(MenuItem mi)
    {
        // 勾选该菜单项
        mi.setChecked(true);  //①
        switch (mi.getItemId())
        {
            case R.id.red:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.RED);
                break;
            case R.id.green:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.GREEN);
                break;
            case R.id.blue:
                mi.setChecked(true);
                txt.setBackgroundColor(Color.BLUE);
                break;
        }
        return true;
    }
    @Override
    // 菜单项被单击后的回调方法
    public boolean onOptionsItemSelected(MenuItem mi)
    {
        if(mi.isCheckable())
        {
            // 勾选该菜单项
            mi.setChecked(true);  //②
        }
        //判断单击的是哪个菜单项，并有针对性地作出响应
        switch (mi.getItemId())
        {
            case R.id.font_10:
                txt.setTextSize(10 * 2);
                break;
            case R.id.font_12:
                txt.setTextSize(12 * 2);
                break;
            case R.id.font_14:
                txt.setTextSize(14 * 2);
                break;
            case R.id.font_16:
                txt.setTextSize(16 * 2);
                break;
            case R.id.font_18:
                txt.setTextSize(18 * 2);
                break;
            case R.id.red_font:
                txt.setTextColor(Color.RED);
                mi.setChecked(true);
                break;
            case R.id.green_font:
                txt.setTextColor(Color.GREEN);
                mi.setChecked(true);
```

```
                break;
            case R.id.blue_font:
                txt.setTextColor(Color.BLUE);
                mi.setChecked(true);
                break;
            case R.id.plain_item:
                Toast toast = Toast.makeText(MenuResTest.this
                    , "您单击了普通菜单项" , Toast.LENGTH_SHORT);
                toast.show();
                break;
        }
        return true;
    }
}
```

上面的程序中两行粗体字代码就是加载选项菜单资源、上下文菜单资源的关键代码。

需要指出，可勾选菜单的勾选状态必须由程序代码来控制，所以上面的程序中①、②号代码自己控制了菜单项勾选状态的切换。这一点是笔者感到很奇怪的地方：为何 Android 系统不为我们处理这些细节？这些细节给我们编程带来了不少烦琐的处理。

从上面的程序可以看出，如果使用 XML 资源文件来定义菜单，就像使用布局文件来定义应用程序界面一样，Android 应用的 Java 代码就会简单很多，因此可维护性更好。

归纳起来，使用 XML 资源定义菜单有如下两个好处。

- XML 资源文件不仅负责定义应用界面，也负责定义菜单，这样可把所有界面相关的内容交给 XML 文件管理，而 Java 代码的功能更集中。
- 后期更新、维护应用时，如果需要更新、维护菜单，打开、编辑 XML 文件即可，避免对 Java 文件的修改。

该应用程序的运行效果与前面介绍的菜单示例的效果大致相同，故此处不再给出。

2.10.7 使用 PopupMenu 创建弹出式菜单

PopupMenu 代表弹出式菜单，它会在指定组件上弹出 PopupMenu，默认情况下，PopupMenu 会显示在该组件的下方或者上方。PopupMenu 可增加多个菜单项，并可为菜单项增加子菜单。

使用 PopupMenu 创建菜单的步骤非常简单，只要如下步骤即可。

① 调用 new PopupMenu(Context context, View anchor)创建下拉菜单，anchor 代表要激发该弹出菜单的组件。

② 调用 MenuInflater 的 inflate()方法将菜单资源填充到 PopupMenu 中。

③ 调用 PopupMenu 的 show()方法显示弹出式菜单。

下面的实例示范了使用 PopupMenu 的功能和用法。下面的实例的界面布局文件中仅有一个普通按钮，界面布局文件非常简单，故此处不再给出代码。

该实例的 Activity 的代码如下。

程序清单：codes\02\2.10\PopupMenuTest\src\org\crazyit\ui\PopupMenuTest.java

```
public class PopupMenuTest extends Activity
{
    PopupMenu popup = null;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void onPopupButtonClick(View button)
    {
        // 创建 PopupMenu 对象
        popup = new PopupMenu(this, button);
        // 将 R.menu.popup_menu 菜单资源加载到 popup 菜单中
        getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());
        // 为 popup 菜单的菜单项单击事件绑定事件监听器
        popup.setOnMenuItemClickListener(
            new PopupMenu.OnMenuItemClickListener()
        {
            @Override
            public boolean onMenuItemClick(MenuItem item)
            {
                switch (item.getItemId())
                {
                    case R.id.exit:
                        // 隐藏该对话框
                        popup.dismiss();
                        break;
                    default:
                        // 使用 Toast 显示用户单击的菜单项
                        Toast.makeText(PopupMenuTest.this,
                            "您单击了【" + item.getTitle() + "】菜单项"
                            , Toast.LENGTH_SHORT).show();
                }
                return true;
            }
        });
        popup.show();
    }
}
```

上面的程序中第一行粗体字代码创建了一个 PopupMenu 对象，第二行粗体字代码指定将该 R.menu.popup_menu 菜单资源文件填充到 PopupMenu 中，这样即可实现当用户单击界面按钮时弹出 Popup 菜单。

运行该程序，单击程序界面上的按钮，将可以看到如图 2.88 所示界面。



图 2.88 使用 PopupMenu 开发弹出式菜单

2.11 使用活动条（ActionBar）

活动条（ActionBar）是 Android 3.0 的重要更新之一。ActionBar 位于传统标题栏的位置，也就是显示的屏幕的顶部。ActionBar 可显示应用的图标和 Activity 标题——也就是前面应用程序的顶部显示的内容。除此之外，ActionBar 的右边还可以显示活动项（Action Item）。

归纳起来，ActionBar 提供了如下功能。

- 显示选项菜单的菜单项（将菜单项显示成 Action Item）。
- 使用程序图标作为返回 Home 主屏或向上的导航操作。
- 提供交互式 View 作为 Action View。
- 提供基于 Tab 的导航方式，可用于切换多个 Fragment。
- 提供基于下拉的导航方式。

2.11.1 启用 ActionBar

Android 3.0版本已经默认启用了ActionBar，因此只要在AndroidManifest.xml文件的SDK配置中指定了该应用的目标版本高于 11（Android 3.0 的版本号），那么默认就会启用ActionBar。例如如下配置：

```
<uses-sdk android:minSdkVersion="10"
    android:targetSdkVersion="17" />
```

指定该应用程序可以部署在Android 4.2平台上，同时兼容Android 2.3.3及更高版本。如果Android版本高于Android 3.0，该应用将会启用ActionBar。

如果希望关闭 ActionBar，可以设置该应用的主题为 Xxx.NoActionBar，例如如下配置片段：

```
<application android:icon="@drawable/ic_launcher"
    android:theme="@android:style/Theme.Holo.NoActionBar"
    android:label="@string/app_name">
    ...
</application>
```

上面的粗体字代码指定该应用关闭 ActionBar 功能。一旦关闭了 ActionBar，该 Android 应用将不能使用 ActionBar。

实际项目中，通常推荐使用代码来控制 ActionBar 的显示、隐藏，ActionBar 提供了如下方法来控制显示、隐藏。

- show()：显示 ActionBar。
- hide()：隐藏 ActionBar。

如下实例示范了如何通过代码来控制 ActionBar 的显示、隐藏。

该实例的界面布局文件中只定义了两个按钮，界面布局文件非常简单，故此处不再给出界面布局文件。该实例的 Activity 代码如下。

程序清单：codes\02\2.11\ActionBarTest\src\org\crazyit\ui\ActionBarTest.java

```
public class ActionBarTest extends Activity
{
    ActionBar actionBar;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取该 Activity 的 ActionBar
        // 只有当应用主题没有关闭 ActionBar 时，该代码才能返回 ActionBar
        actionBar = getActionBar();
    }
    // 为“显示 ActionBar”按钮定义事件处理方法
    public void showActionBar(View source)
    {
        // 显示 ActionBar
        actionBar.show();
    }
    // 为“隐藏 ActionBar”按钮定义事件处理方法
    public void hideActionBar(View source)
    {
        // 隐藏 ActionBar
        actionBar.hide();
```

```
        }
    }
```

上面的程序中第一行粗体字代码调用了 getActionBar()方法获取该 Activity 关联的 ActionBar。接下来就可以调用 ActionBar 的方法来控制它的显示、隐藏。



图 2.89 隐藏 ActionBar

运行该程序并单击“隐藏 ActionBar”按钮，将可以看到如图 2.89 所示界面。

从图 2.89 可以看出，此处程序顶端的 ActionBar 已经“消失”了。

2.11.2 使用 ActionBar 显示选项菜单

前面介绍菜单时已经指出，Android 不再强制要求手机必须提供 MENU 按键，这样可能导致用户无法打开选项菜单。为了解决这个问题，Android 已经提供了 ActionBar 作为解决方案，ActionBar 可以将选项菜单显示成 Action Item。

从 Android 3.0 开始，MenuItem 新增了如下方法。

- setShowAsAction(int actionEnum)：该方法设置是否将该菜单项显示在 ActionBar 上，作为 ActionItem。

该方法支持如下参数值。

- SHOW_AS_ACTION_ALWAYS：总是将该 MenuItem 显示在 ActionBar 上。
- SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW：将该 Action View 折叠成普通菜单项。
- SHOW_AS_ACTION_IF_ROOM：当 ActionBar 位置足够时才显示 MenuItem。
- SHOW_AS_ACTION_NEVER：不将该 MenuItem 显示在 ActionBar 上。
- SHOW_AS_ACTION_WITH_TEXT：将该 MenuItem 显示在 ActionBar 上，并显示该菜单项的文本。

正如前面看到的，实际项目推荐使用 XML 资源文件来定义菜单，因此 Android 允许在 XML 菜单资源文件中为<item.../>元素指定如下属性。

- android:showAsAction：该属性值的作用类似于 setShowAsAction(int actionEnum)方法。因此该属性也能支持类似于上面的属性值。

下面的程序对前面关于菜单的示例稍作修改，只是在定义菜单资源文件时为<item.../>元素增加了 android:showAsAction 属性。下面是本实例的菜单资源文件代码。

程序清单：codes\02\2.11\ActionItemTest\res\menu\my_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="@string/font_size"
        android:showAsAction="always|withText"
        android:icon="@drawable/font">
        <menu>
            <!-- 定义一组单选菜单项 -->
            <group android:checkableBehavior="single">
                <!-- 定义多个菜单项 -->
                <item
                    android:id="@+id/font_10"
                    android:title="@string/font_10"/>
                <item
```

```
                android:id="@+id/font_12"
                android:title="@string/font_12"/>
            <item
                android:id="@+id/font_14"
                android:title="@string/font_14"/>
            <item
                android:id="@+id/font_16"
                android:title="@string/font_16"/>
            <item
                android:id="@+id/font_18"
                android:title="@string/font_18"/>
        </group>
    </menu>
</item>
<!-- 定义一个普通菜单项 -->
<item android:id="@+id/plain_item"
    android:showAsAction="always|withText"
    android:title="@string/plain_item">
</item>
<item android:title="@string/font_color"
    android:showAsAction="always"
    android:icon="@drawable/color">
    <menu>
        <!-- 定义一组允许复选的菜单项 -->
        <group>
            <!-- 定义三个菜单项 -->
            <item
                android:id="@+id/red_font"
                android:title="@string/red_title"/>
            <item
                android:id="@+id/green_font"
                android:title="@string/green_title"/>
            <item
                android:id="@+id/blue_font"
                android:title="@string/blue_title"/>
        </group>
    </menu>
</item>
</menu>
```

上面三行粗体字代码为选项菜单项增加了 android:showAsAction 属性，该属性可控制将这些菜单项显示在 ActionBar 上。再次运行该实例，将可以看到如图 2.90 所示的 ActionItem。

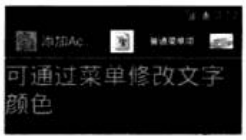


图 2.90　使用 ActionBar 显示选项菜单项

正如图 2.90 所显示的，手机顶部的 ActionBar 的空间是有限的，当选项菜单的菜单项很多时，ActionBar 无法同时显示所有的选项菜单项。Android 将会根据不同手机设备采取不同行为。

- 对于有 MENU 按键的手机，用户单击 MENU 按键即可看到剩余的选项菜单项。
- 对于没有 MENU 按键的手机，ActionBar 会在最后显示一个折叠图标，用户单击该折叠图标将会显示剩余的选项菜单项，类似于以前单击 MENU 按键后出现“More”按钮。

2.11.3　启用程序图标导航

为了将应用程序图标转变成可以点击的图标，可以调用 ActionBar 的如下方法。

- setDisplayHomeAsUpEnabled(boolean showHomeAsUp)：设置是否将应用程序图标转变成可点击的图标，并在图标上添加一个向左的箭头。
- setDisplayOptions(int options)：通过传入 int 类型常量来控制该 ActionBar 的显示选项。
- setDisplayShowHomeEnabled(boolean showHome)：设置是否显示应用程序的图标。
- setHomeButtonEnabled(boolean enabled)：设置是否将应用程序图标转变成可点击的按钮。

下面的程序将该 Activity 的程序图标转变成可点击的图标，并控制用于单击该图标时直接返回程序的主 Activity。该程序界面布局文件、菜单资源文件保持不变。下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionHomeTest\src\org\crazyit\ui\ActionHomeTest.java

```
public class ActionHomeTest extends Activity
{
    private TextView txt;
    ActionBar actionBar;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txt = (TextView) findViewById(R.id.txt);
        actionBar = getActionBar();
        // 设置是否显示应用程序图标
        actionBar.setDisplayShowHomeEnabled(true);
        // 将应用程序图标设置为可点击的按钮
        // actionBar.setHomeButtonEnabled(true);
        // 将应用程序图标设置为可点击的按钮，并在图标上添加向左箭头
        actionBar.setDisplayHomeAsUpEnabled(true);
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater inflator = new MenuInflater(this);
        // 状态 R.menu.context 对应的菜单，并添加到 menu 中
        inflator.inflate(R.menu.my_menu, menu);
        return super.onCreateOptionsMenu(menu);
    }
    @Override
    // 选项菜单的菜单项被单击后的回调方法
    public boolean onOptionsItemSelected(MenuItem mi)
    {
        if(mi.isCheckable())
        {
            mi.setChecked(true);  //②
        }
        // 判断单击的是哪个菜单项，并有针对性地作出响应
        switch (mi.getItemId())
        {
            case android.R.id.home:
                // 创建启动 FirstActivity 的 Intent
                Intent intent = new Intent(this, FirstActivity.class);
                // 添加额外的 Flag，将 Activity 栈中处于 FirstActivity 之上的
                Activity 弹出
```

```
                    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                    // 启动 intent 对应的 Activity
                    startActivity(intent);
                    break;
                ...
            }
            return true;
        }
    }
```

上面的程序中前两行粗体字代码的任意一行都可将 ActionBar 上的应用程序图标转换成可点击的图标。接下来程序为点击事件绑定了事件监听器，当用户单击 ID 为 android.R.id.home 的 Action Item 时（应用程序图标的 ID）时，程序使用 Intent 返回了应用程序主 Activity，如上面的程序中第二段粗体字代码所示。



提示：

上面的实例中还定义了 FirstActivity、SecondActivity 两个 Activity，关于开发、定义 Activity 的详细步骤，请参考本书第 4 章。

2.11.4 添加 Action View

ActionBar 上除了可以显示普通的 Action Item 之外，还可以显示普通的 UI 组件。为了在 ActionBar 上添加 Action View，可以用如下两种方式。

- 定义 Action Item 时使用 android:actionViewClass 属性指定 Action View 的实现类。
- 定义 Action Item 时使用 android:actionLayout 属性指定 Action View 对应的视图资源。



实例："标题"上的时钟

下面的实例将会在菜单资源中定义两个 Action Item，但这两个 Action Item 都是使用 Action View，而不是普通的 Action Item。资源文件代码如下。

程序清单：codes\02\2.11\ActionViewTest\res\menu\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item
      android:id="@+id/search"
      android:title="@string/menu_settings"
      android:orderInCategory="100"
      android:showAsAction="always"
      android:actionViewClass="android.widget.SearchView"/>
<item
      android:id="@+id/progress"
      android:title="@string/menu_settings"
      android:orderInCategory="100"
      android:showAsAction="always"
      android:actionLayout="@layout/clock"/>
</menu>
```

上面的资源文件中两行粗体字代码分别采用两种方式定义了 Action View，这样就可以在 Action Bar 上显示自定义 View。其中第一行粗体字代码指定了 Action View 的实现类为 SearchView；第二行粗体字代码指定该 Action View 对应的界面布局资源为@layout/clock。该

布局资源代码如下。

程序清单：codes\02\2.11\ActionViewTest\res\layout\clock.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<AnalogClock
    xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
/>
```

该界面布局文件仅仅定义了一个 AnalogClock，这表明该 ActionBar 的第二个 Action View 只是一个模拟时钟。

运行该程序，可以看到如图 2.91 所示界面。



图 2.91 添加 Action View

2.11.5 使用 ActionBar 实现 Tab 导航

ActionBar 还有常用的功能：实现 Tab 导航。ActionBar 在顶端生成多个 Tab 标签，当用户点击某个 Tab 标签时，系统根据用户点击事件导航指定 Tab 页面。

为了使用 ActionBar 实现 Tab 导航，按如下步骤进行即可。

① 调用 ActionBar 的 setNavigationMode(ActionBar.NAVIGATION_MODE_TABS)方法设置使用 Tab 导航方式。

② 调用 ActionBar 的 addTab()方法添加多个 Tab 标签，并为每个 Tab 标签添加事件监听器。

实际项目中为了更好地展现 Tab 导航效果，ActionBar 通常会与 Fragment 结合使用，因此这里先简单介绍 Fragment 的用法。

Fragment 是 Android 3.0 新增的重要 API，Fragment 相当于 Activity 片段（Fragment 本来就是片段的意思），我们通常使用单独的 Activity 组合多个 Fragment，这样既可在一个 Activity 创建多个用户界面。除此之外，也可让多个 Activity 复用同一个 Fragment。总之，Fragment 相当于 Activity 的模块化区域。

Fragment 有自己的生命周期，它也可以接收、处理属于它自身的事件，并允许 Activity 运行期间动态地添加、删除 Fragment。

Fragment 允许定义自己的布局，也可通过生命周期回调方法定义自己的行为，这一点 Fragment 非常像 Activity。

与开发 Activity 类似的是，开发者自定义的 Fragment 也需要继承 Fragment，并重写它的生命周期方法，通常会重写 Fragment 的 onCreateView()生命周期方法。



提示：

关于 Fragment 的生命周期及更详细的介绍，请参考本书第 4 章。

下面通过实例来介绍 ActionBar 结合 Fragment 实现 Tab 导航。

实例：ActionBar 结合 Fragment 实现 Tab 导航

该实例的界面布局文件非常简单，该布局文件中只是定义了一个简单的容器，该容器甚至没有太多要求，既可使用 LinearLayout，也可使用 RelativeLayout……甚至是普通 ViewGroup 即可，该容器只是用于盛装 Fragment。该实例的界面布局文件如下。

程序清单：codes\02\2.11\ActionBar_TabNav\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

上面的界面布局文件只是定义了一个 LinearLayout 作为容器，接下来 Activity 将会使用该容器动态盛装 Fragment。下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionBar_TabNav\src\org\crazyit\ui\ActionBar_TabNav.java

```
public class ActionBar_TabNav extends Activity implements
    ActionBar.TabListener
{
    private static final String SELECTED_ITEM = "selected_item";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ActionBar actionBar = getActionBar();
        // 设置 ActionBar 的导航方式：Tab 导航
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
        // 依次添加三个 Tab 页，并为三个 Tab 标签添加事件监听器
        actionBar.addTab(actionBar.newTab().setText("第一页")
            .setTabListener(this));
        actionBar.addTab(actionBar.newTab().setText("第二页")
            .setTabListener(this));
        actionBar.addTab(actionBar.newTab().setText("第三页")
            .setTabListener(this));
    }
    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState)
    {
        if (savedInstanceState.containsKey(SELECTED_ITEM))
        {
            // 选中前面保存的索引对应的 Fragment 页
            getActionBar().setSelectedNavigationItem(
                savedInstanceState.getInt(SELECTED_ITEM));
        }
    }
    @Override
    public void onSaveInstanceState(Bundle outState)
    {
        // 将当前选中的 Fragment 页的索引保存到 Bundle 中
        outState.putInt(SELECTED_ITEM,
            getActionBar().getSelectedNavigationIndex());
    }
    @Override
    public void onTabUnselected(ActionBar.Tab tab,
        FragmentTransaction fragmentTransaction)
    {
    }
    // 当指定 Tab 被选中时激发该方法
    @Override
    public void onTabSelected(ActionBar.Tab tab,
            FragmentTransaction fragmentTransaction)
    {
```

```
            // 创建一个新的 Fragment 对象
            Fragment fragment = new DummyFragment();
            // 创建一个 Bundle 对象，用于向 Fragment 传入参数
            Bundle args = new Bundle();
            args.putInt(DummyFragment.ARG_SECTION_NUMBER,
                    tab.getPosition() + 1);
            // 向 fragment 传入参数
            fragment.setArguments(args);
            // 获取 FragmentTransaction 对象
            FragmentTransaction ft = getFragmentManager()
                    .beginTransaction();
            // 使用 fragment 代替该 Activity 中的 container 组件
            ft.replace(R.id.container, fragment);
            // 提交事务
            ft.commit();
        }
        @Override
        public void onTabReselected(ActionBar.Tab tab,
                FragmentTransaction fragmentTransaction)
        {
        }
    }
```

上面的第一段粗体字代码设置 ActionBar 使用 Tab 导航，为该 ActionBar 添加了三个 Tab 标签，并为每个 Tab 标签都设置了事件监听器。

当用户单击 ActionBar 的指定 Tab 标签时，系统将会激发该监听器的 onTabSelected()方法，因此上面的第二段粗体字代码实现了 onTabSelected()方法，并在该方法中根据用户选中的 Tab 标签替换新的 Fragment。

上面的实例用到了一个 DummyFragment，这是一个简单的 Fragment，它只是显示一个简单的 TextView 组件。下面是该 DummyFragment 的代码。

程序清单：codes\02\2.11\ActionBar_TabNav\src\org\crazyit\ui\DummyFragment.java

```
public class DummyFragment extends Fragment
{
    public static final String ARG_SECTION_NUMBER = "section_number";
    // 该方法的返回值就是该 Fragment 显示的 View 组件
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState)
    {
        TextView textView = new TextView(getActivity());
        textView.setGravity(Gravity.START);
        // 获取创建该 Fragment 时传入的参数 Bundle
        Bundle args = getArguments();
        // 设置 TextView 显示的文本
        textView.setText(args.getInt(ARG_SECTION_NUMBER) + "");
        textView.setTextSize(30);
        // 返回该 TextView
        return textView;
    }
}
```

运行该实例，可以看到如图 2.92 所示效果。

需要指出的是，Android 的最新版本的 ADT 工具已经为 ActionBar 的 Tab 导航提供了支持，当使用 Eclipse 创建 Android 项目时，ADT 的向导工具在创建 Activity 时会看到如图 2.93 所示界面。



图 2.92　ActionBar 结合 Fragment 实现 Tab 导航



图 2.93　为 ActionBar 选择导航方式

从图 2.93 可以看出，ADT 为 Activity 提供了 4 种导航方式，其中 Tabs 导航方式就是刚刚介绍的第一个实例。下面将会详细介绍另外三种导航方式。

实例：Android 3.0 以前的 Fragment 支持

Fragment 非常实用，Android 也为 3.0 以前的平台增加了 Fragment 支持，只是该 Fragment 不是继承 android.app.Fragment，而是继承 android.support.v4.app.Fragment。

除此之外，Android 还为该 android.support.v4.app.Fragment 提供了如下配套类。

- FragmentActivity：在早期版本上使用 Fragment 必须借助于 FragmentActivity 的支持，只有该支持类提供的 getSupportFragmentManager() 方法才能获取 FragmentManager 管理器。
- ViewPager：它是 Fragment 容器，可以同时管理多个 Fragment，并允许多个 Fragment 切换时提供动画效果。
- FragmentPagerAdapter：Adapter 类，用于为 ViewPager 提供多个 Fragment。通常用于被扩展。



提示：

FragmentPagerAdapter 的作用有点类似于前面介绍的 Adapter，只是 Adapter 用于为 AdapterView 提供多个列表项；而 FragmentPagerAdapter 则专门为 ViewPager 提供多个 Fragment。

- PagerTitleStrip：与 ViewPager 结合使用，用于在 ViewPager 上显示“导航条”。

该实例的界面布局文件将会使用 ViewPager 容器，该容器可以盛装多个 Fragment，并为多个 Fragment 切换时提供动画支持。该实例的布局文件如下。

程序清单：codes\02\2.11\ActionBar_TabSwipeNav\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- 定义导航状态条组件 -->
    <android.support.v4.view.PagerTitleStrip
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
```

```
            android:background="#33b5e5"
            android:textColor="#fff"
            android:paddingTop="4dp"
            android:paddingBottom="4dp" />
</android.support.v4.view.ViewPager>
```

上面的布局文件中定义了一个 ViewPager 组件，并为该 ViewPager 组件定义了配套的 PagerTitleStrip 组件——它是一个导航状态条组件。

接下来 Activity 中还是需要按上面介绍的两个步骤来启用 ActionBar 的 Tab 导航支持。除此之外，为了让 ViewPager 组件能正常工作，Activity 需要为该 ViewPager 组件创建并设置 FragmentPagerAdapter。

下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionBar_TabSwipeNav\src\org\crazyit\ui\ActionBar_TabSwipeNav.java

```
public class ActionBar_TabSwipeNav extends FragmentActivity
    implements ActionBar.TabListener
{
    ViewPager viewPager;
    ActionBar actionBar;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 ActionBar 对象
        actionBar = getActionBar();
        // 获取 ViewPager
        viewPager = (ViewPager) findViewById(R.id.pager);
        // 创建一个 FragmentPagerAdapter 对象，该对象负责为 ViewPager 提供多个 Fragment
        FragmentPagerAdapter pagerAdapter = new FragmentPagerAdapter(
                getSupportFragmentManager())
        {
            // 获取第 position 位置的 Fragment
            @Override
            public Fragment getItem(int position)
            {
                Fragment fragment = new DummyFragment();
                Bundle args = new Bundle();
                args.putInt(DummyFragment.ARG_SECTION_NUMBER, position + 1);
                fragment.setArguments(args);
                return fragment;
            }
            // 该方法的返回值 i 表明该 Adapter 总共包括多少个 Fragment
            @Override
            public int getCount()
            {
                return 3;
            }
            // 该方法的返回值决定每个 Fragment 的标题
            @Override
            public CharSequence getPageTitle(int position)
            {
                switch (position)
                {
                    case 0:
                        return "第一页";
                    case 1:
                        return "第二页";
                    case 2:
```

```
                        return "第三页";
                    }
                    return null;
                }
            };
            // 设置 ActionBar 使用 Tab 导航方式
            actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);
            // 遍历 pagerAdapter 对象所包含的全部 Fragment
            // 每个 Fragment 对应创建一个 Tab 标签
            for (int i = 0; i < pagerAdapter.getCount(); i++)
            {
                actionBar.addTab(actionBar.newTab()
                    .setText(pagerAdapter.getPageTitle(i))
                    .setTabListener(this));
            }
            // 为 ViewPager 组件设置 FragmentPagerAdapter
            viewPager.setAdapter(pagerAdapter); //①
            // 为 ViewPager 组件绑定事件监听器
            viewPager.setOnPageChangeListener(
                new ViewPager.SimpleOnPageChangeListener()
                {
                    // 当 ViewPager 显示的 Fragment 发生改变时激发该方法
                    @Override
                    public void onPageSelected(int position)
                    {
                        actionBar.setSelectedNavigationItem(position);
                    }
                });
        }
        @Override
        public void onTabUnselected(ActionBar.Tab tab,
                FragmentTransaction fragmentTransaction)
        {
        }
        // 当指定 Tab 被选中时激发该方法
        @Override
        public void onTabSelected(ActionBar.Tab tab,
                FragmentTransaction fragmentTransaction)
        {
            viewPager.setCurrentItem(tab.getPosition()); //②
        }
        @Override
        public void onTabReselected(ActionBar.Tab tab,
                FragmentTransaction fragmentTransaction)
        {
        }
    }
```

上面的程序采用匿名内部类的形式创建了一个 FragmentPagerAdapter 对象。接下来程序在①号代码处为 ViewPager 组件设置了该 FragmentPagerAdapter 对象，这样即可让该 ViewPager 正常工作。

为了启用 ActionBar 的 Tab 导航支持，上面的粗体字代码同样遵守前面介绍的两个步骤：先调用 ActionBar 的 setNavigationMode(ActionBar.NAVIGATION_MODE_TABS)启用 Tab 导航支持，再调用 ActionBar 的 addTab()方法添加 Tab 标签，并为 Tab 标签绑定事件监听器。

由于此处使用了 ViewPager 来管理多个 Fragment，程序代码处理 Fragment 的切换时更简单：只要调用 ViewPager 的 setCurrentItem()方法来显示指定 Fragment 即可。如以上程序中②号代码所示。

运行上面的程序，可以看到如图 2.94 所示界面。

当用户单击图 2.94 所示界面上的 Tab 标签，或通过拖动来切换 ViewPager 显示的 Fragment 时，将可以看到 Fragment 切换时的动画效果。



图 2.94 Android 3.0 以前的 Fragment 支持

2.11.6 使用 ActionBar 实现下拉式导航

ActionBar 除可提供 Tab 导航支持之外，还提供了下拉式（DropDown）导航方式。下拉式导航的 ActionBar 在顶端生成下拉列表框，当用户单击某个列表项时，系统根据用户单击事件导航指定 Fragment。

为了使用 ActionBar 实现 Tab 导航，按如下步骤进行即可。

① 调用 ActionBar 的 actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST)方法设置使用下拉列表的导航方式。

② 调用 ActionBar 的 setListNavigationCallbacks(SpinnerAdapter adapter, ActionBar.OnNavigationListener callback) ()添加多个列表项，并为每个列表项设置事件监听器。其中第一个参数 Adapter 负责提供多个列表项，第二个参数为事件监听器。

实例：ActionBar 结合 Fragment 实现下拉式导航

下面的实例的布局文件与前面介绍的 Tab 导航实例的布局文件完全相同：该布局文件只要有一个简单的容器即可，该容器用于盛装多个 Fragment。此处不再给出该界面布局文件。

下面是该 Activity 的代码。

程序清单：codes\02\2.11\ActionBar_DropDownNav\src\org\crazyit\ui\ActionBar_DropDownNav. java

```
public class ActionBar_DropDownNav extends Activity implements
        ActionBar.OnNavigationListener
{
    private static final String SELECTED_ITEM = "selected_item";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ActionBar actionBar = getActionBar();
        // 设置 ActionBar 是否显示标题
        actionBar.setDisplayShowTitleEnabled(true);
        // 设置导航模式，使用 List 导航
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
        // 为 actionBar 安装 ArrayAdapter
        actionBar.setListNavigationCallbacks(
            new ArrayAdapter<String>(ActionBar_DropDownNav.this,
                android.R.layout.simple_list_item_1,
                android.R.id.text1, new String[]
                {"第一页","第二页","第三页" }), this);
    }
    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState)
    {
        if (savedInstanceState.containsKey(SELECTED_ITEM))
        {
            // 选中前面保存的索引对应的 Fragment 页
            getActionBar().setSelectedNavigationItem(
                savedInstanceState.getInt(SELECTED_ITEM));
```

```
        }
    }
    @Override
    public void onSaveInstanceState(Bundle outState)
    {
        // 将当前选中的 Fragment 页的索引保存到 Bundle 中
        outState.putInt(SELECTED_ITEM,
            getActionBar().getSelectedNavigationIndex());
    }
    // 当导航项被选中时激发该方法
    @Override
    public boolean onNavigationItemSelected(int position, long id)
    {
        // 创建一个新的 Fragment 对象
        Fragment fragment = new DummyFragment();
        // 创建一个 Bundle 对象，用于向 Fragment 传入参数
        Bundle args = new Bundle();
        args.putInt(DummyFragment.ARG_SECTION_NUMBER, position + 1);
        // 向 fragment 传入参数
        fragment.setArguments(args);
        // 获取 FragmentTransaction 对象
        FragmentTransaction ft = getFragmentManager().beginTransaction();
        // 使用 fragment 代替该 Activity 中的 container 组件
        ft.replace(R.id.container, fragment);
        // 提交事务
        ft.commit();
        return true;
    }
}
```

上面的程序中第一段粗体字代码就是为 ActionBar 启用下拉导航支持的关键代码，这段代码做了上面介绍的两件事情：先调用 ActionBar 的 setNavigationMode(ActionBar.NAVIGATION_MODE_LIST)启用下拉列表导航支持；然后为 ActionBar 传入 ArrayAdapter（当然也可使用 SimpleAdapter 或扩展 BaseAdapter 的对象）和监听器即可。

当用户选中指定的导航项时，将会激发该监听器的 onNavigationItemSelected()，该方法的处理逻辑与前面 Tab 导航实例中 onTabSelected()处理方法的处理逻辑完全相同，该实例所使用的 DummyFragment 与前面的 Fragment 类的代码也完全相同，此处不再赘述。

运行该实例，可以看到如图 2.95 所示界面。



图 2.95　ActionBar 结合 Fragment 实现下拉式导航

2.12　本章小结

本章主要介绍了 Android 应用界面开发的相关知识，对于一个手机应用来说，它面临的最终用户都是不太懂软件的普通人，这批用户第一眼看到的就是软件界面，因此为 Android 系统提供一个友好的用户界面十分重要。学习本章需要重点掌握 View 与 ViewGroup 的功能和用法；Android 系统所有的基本 UI 组件、高级 UI 组件也都需要重点掌握。除此之外，用户界面少不了需要对话框与菜单，Android 为对话框提供了 AlertDialog 类，还提供了 PopupWindow、DatePickerDialog、TimePickerDialog 用于辅助开发对话框。另外，Toast 是手机系统特有的一种“准对话框”，它可用于显示简单的提示信息，而且一段时间后会自动隐藏，非常方便。Android 为菜单支持提供了 SubMenu、ContextMenu、MenuItem 等 API，读者必须掌握这些 API 的用法，并能通过它们为 Android 应用添加菜单支持。

CHAPTER

3

第3章 Android的事件处理

本章要点

- 事件处理概述与 Android 事件处理
- 基于监听的事件处理模型
- 事件与事件监听器接口
- 实现事件监听器的方式
- 基于回调的事件处理模型
- 基于回调的事件传播
- 常见的事件回调方法
- 响应系统设置事件
- 重写 onConfigurationChanged 方法响应系统设置更改
- Handler 类功能与用法
- 使用 Handler 更新程序界面
- Handler、Looper、MessageQueue 工作原理
- 异步任务的功能与用法

与界面编程紧密相关的知识就是事件处理了，当用户在程序界面上执行各种操作时，应用程序必须为用户动作提供响应动作，这种响应动作就需要通过事件处理来完成。因此本章知识与上一章的内容衔接得非常紧密，实际上我们在介绍上一章示例时已经使用过 Android 的事件处理了。

Android 提供了两种方式的事件处理：基于回调的事件处理和基于监听器的事件处理。熟悉传统图形界面编程的读者对于基于回调的事件处理可能比较熟悉；熟悉 AWT/Swing 开发方式的读者对于基于监听器的事件处理可能比较熟悉。Android 系统充分利用了两种事件处理方式的优点，允许开发者采用自己熟悉的事件处理方式来为用户操作提供响应动作。

本章将会详细介绍 Android 事件处理的各种实现细节，学完本章内容之后，再结合上一章的内容，读者将可以开发出界面友好、人机交互良好的 Android 应用。

3.1 Android 事件处理概述

不管是桌面应用还是手机应用程序，面对最多的就是用户，经常需要处理的就是用户动作——也就是需要为用户动作提供响应，这种为用户动作提供响应的机制就是事件处理。

Android 提供了强大的事件处理机制，包括两套事件处理机制：

- 基于监听的事件处理。
- 基于回调的事件处理。

对于 Android 基于监听的事件处理而言，主要做法就是为 Android 界面组件绑定特定的事件监听器，上一章我们已经见到大量这种事件处理的示例。



提示：

Android 还允许在界面布局文件中为 UI 组件的 android:onClick 属性指定事件监听方法，通过这种方式指定事件监听方法时，开发者需要在 Activity 中定义该事件监听方法（该方法必须有一个 View 类型的形参，该形参代表被单击的 UI 组件），当用户单击该 UI 组件时，系统将会激发 android:onClick 属性所指定的方法。

对于 Android 基于回调的事件处理而言，主要做法就是重写 Android 组件特定的回调方法，或者重写 Activity 的回调方法。Android 为绝大部分界面组件都提供了事件响应的回调方法，开发者只要重写它们即可。

一般来说，基于回调的事件处理可用于处理一些具有通用性的事件，基于回调的事件处理代码会显得比较简洁。但对于某些特定的事件，无法使用基于回调的事件处理，只能采用基于监听的事件处理。

3.2 基于监听的事件处理

基于监听的事件处理是一种更“面向对象”的事件处理，这种处理方式与 Java 的 AWT、Swing 的处理方式几乎完全相同。如果开发者有 AWT、Swing 事件处理的编程经验，基本上可以直接上手编程，甚至不需要学习。如果以前没有任何事件处理的编程经验，就需要花点时间先去理解事件监听的处理模型。

3.2.1 监听的处理模型

在事件监听的处理模型中，主要涉及如下三类对象。

- Event Source（事件源）：事件发生的场所，通常就是各个组件，例如按钮、窗口、菜单等。
- Event（事件）：事件封装了界面组件上发生的特定事情（通常就是一次用户操作）。如果程序需要获得界面组件上所发生事件的相关信息，一般通过 Event 对象来取得。
- Event Listener（事件监听器）：负责监听事件源所发生的事件，并对各种事件做出相应的响应。



提示：

有过 JavaScript、Visual Basic 等编程经验的读者都知道，事件响应的动作实际上就是一系列程序语句，通常以方法的形式组织起来。但 Java 是面向对象的编程语言，方法不能独立存在，所以必须以类的形式来组织这些方法，所以事件监听器的核心就是它所包含的方法——这些方法也被称为事件处理器（Event Handler）。

当用户按下一个按钮或者单击某个菜单项时，这些动作就会激发一个相应的事件，该事件就会触发事件源上注册的事件监听器（特殊的 Java 对象），事件监听器调用对应的事件处理器（事件监听器里的实例方法）来做出相应的响应。

Android 的事件处理机制是一种委派式（Delegation）事件处理方式：普通组件（事件源）将整个事件处理委托给特定的对象（事件监听器）；当该事件源发生指定的事件时，就通知所委托的事件监听器，由事件监听器来处理这个事件。

每个组件均可以针对特定的事件指定一个事件监听器，每个事件监听器也可监听一个或多个事件源。因为同一个事件源上可能发生多种事件，委派式事件处理方式可以把事件源上所有可能发生的事件分别授权给不同的事件监听器来处理；同时也可以让一类事件都使用同一个事件监听器来处理。



提示：

委派式事件处理方式明显“抄袭”了人类社会的分工协作，例如某个单位发生了火灾，该单位通常不会自己处理该事件，而是将该事件委派给消防局（事件监听器）处理；如果发生了打架斗殴事件，则委派给公安局（事件监听器）处理；而消防局、公安局也会同时监听多个单位的火灾、打架斗殴事件。这种委派式的处理方式将事件源和事件监听器分离，从而提供更好的程序模型，有利于提高程序的可维护性。

如图 3.1 所示为事件处理流程示意图。

下面以一个简单的入门程序来示范简单的基于监听的事件处理模型。

先看本程序的界面布局代码，该界面布局中只是定义了两个组件：一个文本框和一个按钮，界面布局代码如下。

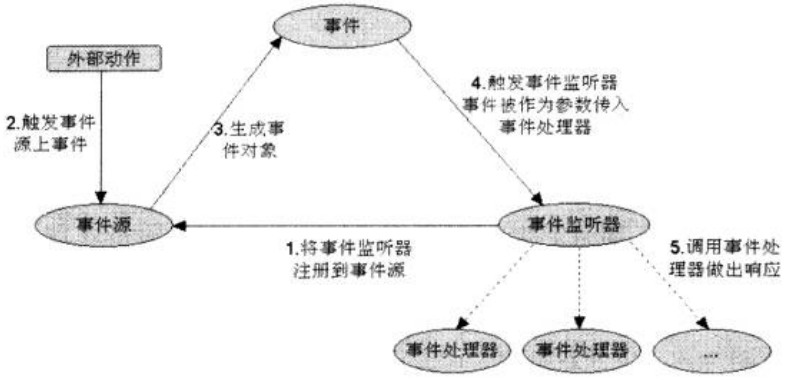


图 3.1　事件处理示意图

程序清单：codes\03\3.2\EventQs\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<EditText
      android:id="@+id/txt"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:editable="false"
      android:cursorVisible="false"
      android:textSize="12pt"/>
<!-- 定义一个按钮，该按钮将作为事件源 -->
<Button
      android:id="@+id/bn"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="单击我"/>
</LinearLayout>
```

上面的程序中定义的按钮将会作为事件源，接下来程序将会为该按钮绑定一个事件监听器——监听器类必须由开发者来实现。下面是 Java 程序代码。

程序清单：codes\03\3.2\EventQs\src\org\crazyit\event\EventQs.java

```
public class EventQs extends Activity
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取应用程序中的 bn 按钮
            Button bn = (Button) findViewById(R.id.bn);
            // 为按钮绑定事件监听器
            bn.setOnClickListener(new MyClickListener()); // ①
      }
      // 定义一个单击事件的监听器
      class MyClickListener implements View.OnClickListener
      {
            // 实现监听器类必须实现的方法，该方法将会作为事件处理器
```

```
        @Override
        public void onClick(View v)
        {
            EditText txt = (EditText) findViewById(R.id.txt);
            txt.setText("bn 按钮被单击了！");
        }
    }
}
```

上面的程序中粗体字代码定义了一个View.OnClickListener实现类，这个实现类将会作为事件监听器使用。程序中①号代码用于为bn按钮注册事件监听器。当程序中的bn按钮被单击时，该处理器被触发，将看到程序中文本框内变为“bn按钮被单击了”。

从上面的程序中可以看出，基于监听的事件处理模型的编程步骤如下。

1. 获取普通界面组件（事件源），也就是被监听的对象。
2. 实现事件监听器类，该监听器类是一个特殊的Java类，必须实现一个XxxListener接口。
3. 调用事件源的setXxxListener方法将事件监听器对象注册给普通组件（事件源）。

当事件源上发生指定事件时，Android会触发事件监听器，由事件监听器调用相应的方法（事件处理器）来处理事件。

把上面的程序与图3.1结合起来看，可以发现基于监听的事件处理有如下规则。

- 事件源：就是程序中的bn按钮，其实开发者不需要太多的额外处理。应用程序中任何组件都可作为事件源。
- 事件监听器：就是程序中的MyClickListener类。监听器类必须由程序员负责实现，实现监听器类的关键就是实现处理器方法。
- 注册监听器：只要调用事件源的setXxxListener(XxxListener)方法即可。

上面三件事情，事件源可以是任何界面组件，不太需要开发者参与；注册监听器也只要一行代码即可，因此事件编程的关键就是实现事件监听器类。

3.2.2 事件和事件监听器

从图3.1中可以看出：当外部动作在Android组件上执行操作时，系统会自动生成事件对象，这个事件对象会作为参数传给事件源上注册的事件监听器。

事件监听的处理模型涉及三个成员：事件源、事件和事件监听器，其中事件源最容易创建，任意界面组件都可作为事件源；事件的产生无须程序员关心，它是由系统自动产生的；所以，实现事件监听器是整个事件处理的核心。

但在上面的程序中，我们并未发现事件的踪迹，这是什么原因呢？这是因为Android对事件监听模型做了进一步简化：如果事件源触发的事件足够简单、事件里封装的信息比较有限，那就无须封装事件对象、将事件对象传入事件监听器。

但对于键盘事件、触摸屏事件等，此时程序需要获取事件发生的详细信息：例如键盘事件需要获取是哪个键触发的事件；触摸屏事件需要获取事件发生的位置等，对于这种包含更多信息的事件，Android同样会将事件信息封装成XxxEvent对象，并把该对象作为参数传入事件处理器。

实例：控制飞机移动

下面以一个简单的飞机游戏为例来介绍键盘事件的监听。游戏中的飞机会随用户单击键

的动作而移动：单击不同的键，飞机向不同方向移动。

为了实现该程序，先开发一个自定义 View，该 View 负责绘制游戏的飞机，该 View 类的代码如下。

程序清单：codes\03\3.2\plane\src\org\crazyit\event\PlainView.java

```
public class PlaneView extends View
{
    public float currentX;
    public float currentY;
    Bitmap plane;
    public PlaneView(Context context)
    {
        super(context);
        // 定义飞机图片
        plane = BitmapFactory.decodeResource(context.getResources(),
                R.drawable.plane);
        setFocusable(true);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 创建画笔
        Paint p = new Paint();
        // 绘制飞机
        canvas.drawBitmap(plane, currentX, currentY, p);
    }
}
```

上面的 PlaneView 足够简单，因为这个程序只需要绘制玩家自己控制的飞机，没有增加“敌机”，所以比较简单。如果游戏中还需要增加“敌机”，那么还需要增加数据来控制敌机的坐标，并会在 View 上绘制敌机。

该游戏几乎不需要页面布局，该游戏直接使用 PlaneView 作为 Activity 显示的内容，并为该 PlaneView 增加键盘事件监听器即可。下面是该程序的 Activity 代码。

程序清单：codes\03\3.2\plane\src\org\crazyit\event\PlaneGame.java

```
public class PlaneGame extends Activity
{
    // 定义飞机的移动速度
    private int speed = 10;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 去掉窗口标题
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // 全屏显示
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
        // 创建 PlaneView 组件
        final PlaneView planeView = new PlaneView(this);
        setContentView(planeView);
        planeView.setBackgroundResource(R.drawable.back);
        // 获取窗口管理器
        WindowManager windowManager = getWindowManager();
        Display display = windowManager.getDefaultDisplay();
        DisplayMetrics metrics = new DisplayMetrics();
        // 获得屏幕宽和高
```

```
        display.getMetrics(metrics);
        // 设置飞机的初始位置
        planeView.currentX = metrics.widthPixels / 2;
        planeView.currentY = metrics.heightPixels - 40;
        // 为draw组件键盘事件绑定监听器
        planeView.setOnKeyListener(new OnKeyListener()
        {
            @Override
            public boolean onKey(View source, int keyCode, KeyEvent event)
            {
                // 获取由哪个键触发的事件
                switch (event.getKeyCode())
                {
                    // 控制飞机下移
                    case KeyEvent.KEYCODE_S:
                        planeView.currentY += speed;
                        break;
                    // 控制飞机上移
                    case KeyEvent.KEYCODE_W:
                        planeView.currentY -= speed;
                        break;
                    // 控制飞机左移
                    case KeyEvent.KEYCODE_A:
                        planeView.currentX -= speed;
                        break;
                    // 控制飞机右移
                    case KeyEvent.KEYCODE_D:
                        planeView.currentX += speed;
                        break;
                }
                // 通知planeView组件重绘
                planeView.invalidate();
                return true;
            }
        });
    }
}
```

上面的程序中粗体字代码就是控制飞机移动的关键代码——由于程序需要根据用户按下的键来确定飞机的移动方向，所以上面的程序先调用了KeyEvent（事件对象）的getKeyCode()来获取触发事件的键，然后针对不同的键来改变游戏中飞机的坐标。

正如前面提到的：如果事件发生时有比较多的信息需要传给事件监听器，那么就需要将事件信息封装成Event对象，该Event对象将作为参数传入事件处理函数。

运行上面的程序，将看到如图3.2所示界面。

对于图3.2所示的“游戏”，当用户按下模拟器右边的4个方向键时，将可以看到“游戏”中的飞机可以上、下、左、右自由移动。



图3.2 控制飞机的移动

在基于事件监听的处理模型中，事件监听器必须实现事件监听器接口，Android为不同的界面组件提供了不同的监听器接口，这些接口通常以内部类的形式存在。以View类为例，它包含了如下几个内部接口。

- View.OnClickListener：单击事件的事件监听器必须实现的接口。
- View.OnCreateContextMenuListener：创建上下文菜单事件的事件监听器必须实现的接口。
- View.onFocusChangeListener：焦点改变事件的事件监听器必须实现的接口。

- View.OnKeyListener：按键事件的事件监听器必须实现的接口。



提示：

上面这个“游戏”还只是一个“雏型”，为了增加这个游戏的可玩性，可以考虑为游戏随机地增加“敌机”，并让“敌机”在屏幕上移动。如果想增加对战效果，还可以考虑增加一个键盘监听器：监听特定按键，特定按键激发飞机射出子弹。

- View.OnLongClickListener：长单击事件的事件监听器必须实现的接口。
- View.OnTouchListener：触摸屏事件的事件监听器必须实现的接口。



提示：

实际上可以把事件处理模型简化成如下理解。当事件源组件上发生事件时，系统将会执行该事件源组件上监听器的对应处理方法。与普通 Java 方法调用不同的是：普通 Java 程序里的方法是由程序主动调用的，事件处理中的事件处理器方法是由系统负责调用的。

通过上面的介绍不难看出，所谓事件监听器，其实就是实现了特定接口的 Java 类的实例。在程序中实现事件监听器，通常有如下几种形式。

- 内部类形式：将事件监听器类定义成当前类的内部类。
- 外部类形式：将事件监听器类定义成一个外部类。
- Activity 本身作为事件监听器类：让 Activity 本身实现监听器接口，并实现事件处理方法。
- 匿名内部类形式：使用匿名内部类创建事件监听器对象。

3.2.3 内部类作为事件监听器类

前面两个程序中所使用的事件监听器类都是内部类形式，使用内部类可以在当前类中复用该监听器类；因为监听器类是外部类的内部类，所以可以自由访问外部类的所有界面组件。这也是内部类的两个优势。

使用内部类来定义事件监听器类的例子可以参看前面的例子程序，此处不再赘述。

3.2.4 外部类作为事件监听器类

使用外部类定义事件监听器类的形式比较少见，主要因为如下两个原因：

- 事件监听器通常属于特定的 GUI 界面，定义成外部类不利于提高程序的内聚性。
- 外部类形式的事件监听器不能自由访问创建 GUI 界面的类中的组件，编程不够简洁。

但如果某个事件监听器确实需要被多个 GUI 界面所共享，而且主要是完成某种业务逻辑的实现，则可以考虑使用外部类的形式来定义事件监听器类。下面的程序定义了一个外部类作为 OnLongClickListener 类，该事件监听器实现了发送短信的功能。

程序清单：codes\03\3.2\SendSms\src\org\crazyit\event\SendSmsListener.java

```
public class SendSmsListener implements OnLongClickListener
{
    private Activity act;
```

```
    private EditText address;
    private EditText content;
    public SendSmsListener(Activity act, EditText address
        , EditText content)
    {
        this.act = act;
        this.address = address;
        this.content = content;
    }
    @Override
    public boolean onLongClick(View source)
    {
        String addressStr = address.getText().toString();
        String contentStr = content.getText().toString();
        // 获取短信管理器
        SmsManager smsManager = SmsManager.getDefault();
        // 创建发送短信的 PendingIntent
        PendingIntent sentIntent = PendingIntent.getBroadcast(act
            , 0, new Intent(), 0);
        // 发送文本短信
        smsManager.sendTextMessage(addressStr, null, contentStr
            , sentIntent, null);
        Toast.makeText(act, "短信发送完成", Toast.LENGTH_LONG).show();
        return false;
    }
}
```

上面的事件监听器类没有与任何 GUI 界面耦合，创建该监听器对象时需要传入两个 EditText 对象和一个 Activity 对象，其中一个 EditText 用于作为收信人号码，一个 EditText 用于作为短信内容。



提示：

上面的程序中3行粗体字代码调用了 SmsManager、PendingIntent 来发送短信，关于 SmsManager、Intent 的用法可参看本书后面的内容。

该程序的界面布局比较简单，故不给出界面布局文件，该程序的 Java 代码如下。

程序清单：codes\03\3.2\SendSms\src\org\crazyit\event\SendSms.java

```
public class SendSms extends Activity
{
    EditText address;
    EditText content;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取页面中收件人地址、短信内容
        address = (EditText)findViewById(R.id.address);
        content = (EditText)findViewById(R.id.content);
        Button bn = (Button)findViewById(R.id.send);
        bn.setOnLongClickListener(new SendSmsListener(
            this , address, content));
    }
}
```

上面的程序中粗体字代码用于为指定按钮的长单击事件绑定监听器，当用户长单击界面中的 bn 按钮时，程序将会触发 SendSmsListener 监听器，该监听器里包含的事件处理方法将

会向指定手机发送短信。

运行上面的程序，将看到如图 3.3 所示界面。

此处使用模拟器来发送短信——不要指望能给你的女朋友发送短信，否则中国移动要恼火了。该程序只能向另一台模拟器发送短信，比如执行如下命令来启动另一台模拟器（直接使用 abc 镜像文件来启动模拟器）：

```
emulator -data abc
```

启动完成后可以在模拟器窗口的标题上看到该模拟器的号码，如图 3.4 所示。

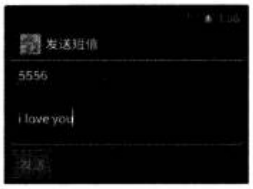


图 3.3　向指定手机发送短信



图 3.4　模拟器的号码

启动了如图 3.4 所示的模拟器之后，就可以通过图 3.3 所示的程序向指定模拟器发送短信了。

注意：

实际上不推荐将业务逻辑实现写在事件监听器中，包含业务逻辑的事件监听器将导致程序的显示逻辑和业务逻辑耦合，从而增加程序后期的维护难度。如果确实有多个实现监听器需要实现相同的业务逻辑功能，可以考虑使用业务逻辑组件来定义业务逻辑功能，再让事件监听器来调用业务逻辑组件的业务逻辑方法。



3.2.5　Activity 本身作为事件监听器

这种形式使用 Activity 本身作为监听器类，可以直接在 Activity 类中定义事件处理器方法，这种形式非常简洁。但这种做法有两个缺点：

- 这种形式可能造成程序结构混乱，Activity 的主要职责应该是完成界面初始化工作，但此时还需包含事件处理器方法，从而引起混乱。
- 如果 Activity 界面类需要实现监听器接口，让人感觉比较怪异。

下面的程序使用 Activity 对象作为事件监听器。

程序清单：codes\03\3.2\ActivityListener\src\org\crazyit\event\ActivityListener.java

```
// 实现事件监听器接口
public class ActivityListener extends Activity
    implements OnClickListener
{
    EditText show;
    Button bn;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (EditText) findViewById(R.id.show);
```

```
        bn = (Button) findViewById(R.id.bn);
        // 直接使用Activity作为事件监听器
        bn.setOnClickListener(this);
    }
    // 实现事件处理方法
    @Override
    public void onClick(View v)
    {
        show.setText("bn按钮被单击了！");
    }
}
```

上面的程序让 Activity 类实现了 OnClickListener 事件监听接口，从而可以在该 Activity 类中直接定义事件处理器方法：onClick(View v)（如上面的粗体字代码所示）。当为某个组件添加该事件监听器对象时，直接使用 this 作为事件监听器对象即可。

3.2.6 匿名内部类作为事件监听器类

大部分时候，事件处理器都没有什么复用价值（可复用代码通常都被抽象成了业务逻辑方法），因此大部分事件监听器只是临时使用一次，所以使用匿名内部类形式的事件监听器更合适。实际上，这种形式是目前使用最广泛的事件监听器形式。下面的程序使用匿名内部类来创建事件监听器。

程序清单：codes\03\3.2\AnonymousListener\src\org\crazyit\event\AnonymousListener.java

```
public class AnonymousListener extends Activity
{
    EditText show;
    Button bn;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (EditText) findViewById(R.id.show);
        bn = (Button) findViewById(R.id.bn);
        // 直接使用Activity作为事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            // 实现事件处理方法
            @Override
            public void onClick(View v)
            {
                show.setText("bn按钮被单击了！");
            }
        });
    }
}
```

上面的程序中粗体字部分使用匿名内部类创建了一个事件监听器对象，“new 监听器接口”或“new 事件适配器”的形式就是用于创建匿名内部类形式的事件监听器。

对于使用匿名内部类作为监听器的形式来说，唯一的缺点就是匿名内部类的语法有点不易掌握，如果读者 Java 基础扎实，匿名内部类的语法掌握较好，通常建议使用匿名内部类作为监听器。

3.2.7 直接绑定到标签

Android 还有一种更简单的绑定事件监听器的方式，直接在界面布局文件中为指定标签绑定事件处理方法。

对于很多 Android 界面组件标签而言，它们都支持 onClick 属性，该属性的属性值就是一个形如 xxx(View source)的方法的方法名。例如如下界面布局文件：

程序清单：codes\03\3.2\bindingTag\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<EditText
      android:id="@+id/show"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:editable="false"
      android:cursorVisible="false"
      />
<!-- 在标签中为按钮绑定事件处理方法 -->
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="单击我"
      android:onClick="clickHandler"
      />
</LinearLayout>
```

上面的程序中粗体字代码用于在界面布局文件中为 Button 按钮绑定一个事件处理方法：clickHanlder，这就意味着开发者需要在该界面布局对应的 Activity 中定义一个 void clickHandler(View source)方法，该方法将会负责处理该按钮上的单击事件。下面是该界面布局对应的 Java 代码。

程序清单：codes\03\3.2\bindingTag\src\org\crazyit\event\BindingTag.java

```
public class BindingTag extends Activity
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
      }
      // 定义一个事件处理方法
      // 其中 source 参数代表事件源
      public void clickHandler(View source)
      {
            EditText show = (EditText) findViewById(R.id.show);
            show.setText("bn 按钮被单击了");
      }
}
```

上面的程序中粗体字代码定义了一个 clickHandler(View source)方法，当程序中的 bn 按钮被单击时，该方法将会被激发并处理 bn 按钮上的单击事件。

3.3 基于回调的事件处理

除了前面介绍的基于监听的事件处理模型之外，Android 还提供了一种基于回调的事件处理模型。从代码实现的角度来看，基于回调的事件处理模型更加简单。

3.3.1 回调机制与监听机制

如果说事件监听机制是一种委托式的事件处理，那么回调机制则恰好与之相反：对于基于回调的事件处理模型来说，事件源与事件监听器是统一的，或者说事件监听器完全消失了。当用户在 GUI 组件上激发某个事件时，组件自己特定的方法将会负责处理该事件。

为了使用回调机制类处理 GUI 组件上所发生的事件，我们需要为该组件提供对应的事件处理方法——而 Java 又是一种静态语言，我们无法为某个对象动态地添加方法，因此只能继承 GUI 组件类，并重写该类的事件处理方法来实现。

为了实现回调机制的事件处理，Android 为所有 GUI 组件都提供了一些事件处理的回调方法，以 View 为例，该类包含如下方法。

- boolean onKeyDown(int keyCode, KeyEvent event)：当用户在该组件上按下某个按键时触发该方法。
- boolean onKeyLongPress(int keyCode, KeyEvent event)：当用户在该组件上长按某个按键时触发该方法。
- boolean onKeyShortcut(int keyCode, KeyEvent event)：当一个键盘快捷键事件发生时触发该方法。
- boolean onKeyUp(int keyCode, KeyEvent event)：当用户在该组件上松开某个按键时触发该方法。
- boolean onTouchEvent(MotionEvent event)：当用户在该组件上触发触摸屏事件时触发该方法。
- boolean onTrackballEvent(MotionEvent event)：当用户在该组件上触发轨迹球屏事件时触发该方法。

下面的程序示范了基于回调的事件处理机制，正如前面提到的，基于回调的事件处理机制可通过自定义 View 来实现，自定义 View 时重写该 View 的事件处理方法即可。下面是一个自定义按钮的实现类。

程序清单：codes\03\3.3\CallbackHandler\src\org\crazyit\event\MyButton.java

```
public class MyButton extends Button
{
    public MyButton(Context context, AttributeSet set)
    {
        super(context, set);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        super.onKeyDown(keyCode, event);
        Log.v("-crazyit.org-", "the onKeyDown in MyButton");
        // 返回 true，表明该事件不会向外扩散
        return true;
    }
}
```

在上面自定义的 MyButton 类中，我们重写了 Button 类的 onKeyDown(int keyCode, KeyEvent event)方法，该方法将会负责处理按钮上的键盘事件。

接下来在界面布局文件中使用这个自定义 View，界面布局文件如下所示。

程序清单：codes\03\3.3\CallbackHandler\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      >
<!-- 使用自定义 View 时应使用全限定类名 -->
<org.crazyit.event.MyButton
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="单击我"
      />
</LinearLayout>
```

上面的程序中粗体字代码在 XML 界面布局文件中使用 MyButton 组件，接下来 Java 程序无须为该按钮绑定事件监听器——因为该按钮自己重写了 onKeyDown(int keyCode, KeyEvent event)方法，这意味着该按钮将会自己处理相应的事件。

运行上面的程序，先把焦点定位到该按钮上，如图 3.5 所示。

接着单击模拟器上的任意按键，将可以看到 DDMS 的 LogCat 上有如图 3.6 所示的输出。



图 3.5　按钮获得焦点

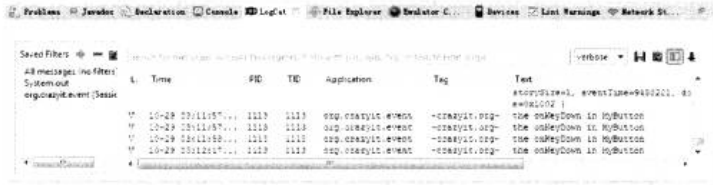


图 3.6　基于回调的事件处理

通过上面的介绍不难发现：对于基于监听的事件处理模型来说，事件源和事件监听器是分离的，当事件源上发生特定事件之后，该事件交给事件监听器负责处理；对于基于回调的事件处理模型来说，事件源和事件监听器是统一的，当事件源发生特定事件之后，该事件还是由事件源本身负责处理。

3.3.2　基于回调的事件传播

几乎所有基于回调的事件处理方法都有一个 boolean 类型的返回值，该返回值用于标识该处理方法是否能完全处理该事件：

- 如果处理事件的回调方法返回 **true**，表明该处理方法已完全处理该事件，该事件不会传播出去。
- 如果处理事件的回调方法返回 **false**，表明该处理方法并未完全处理该事件，该事件会传播出去。

对于基于回调的事件传播而言，某组件上所发生的事情不仅激发该组件上的回调方法，也会触发该组件所在 Activity 的回调用法——只要事件能传播到该 Activity。

下面的一个程序示范了 Android 系统中的事件传播，该程序重写了 Button 类的

onKeyDown(int keyCode, KeyEvent event)方法，而且重写了该 Button 所在 Activity 的 onKeyDown(int keyCode, KeyEvent event)方法——而且程序没有阻止事件传播，因此程序可以看到事件从 Button 传播到 Activity 的情形。

下面是从 Button 派生而出的 MyButton 子类代码。

程序清单：codes\03\3.3\Propagation\src\org\crazyit\event\MyButton.java

```
public class MyButton extends Button
{
    public MyButton(Context context , AttributeSet set)
    {
        super(context , set);
    }
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        super.onKeyDown(keyCode , event);
        Log.v("-MyButton-" , "the onKeyDown in MyButton");
        // 返回 false，表明并未完全处理该事件，该事件依然向外扩散
        return false;
    }
}
```

上面的 MyButton 子类重写了 onKeyDown(int keyCode , KeyEvent event)方法，当用户在该按钮上按下某个键时将会触发该方法。但由于该方法返回了 false，这意味着该事件还会继续向外传播。

该程序也按前一个示例的方式使用该自定义组件，并在 Activity 中重写 public boolean onKeyDown(int keyCode, KeyEvent event)方法，该方法也会在某个按键被按下时被回调。

看如下 Activity 类代码。

程序清单：codes\03\3.3\Propagation\src\org\crazyit\event\Propagation.java

```
public class Propagation extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 绑定事件监听器
        bn.setOnKeyListener(new OnKeyListener()
        {
            @Override
            public boolean onKey(View source
                    , int keyCode, KeyEvent event)
            {
                // 只处理按下键的事件
                if (event.getAction() == KeyEvent.ACTION_DOWN)
                {
                    Log.v("-Listener-", "the onKeyDown in Listener");
                }
                // 返回 false，表明该事件会向外传播
                return false; // ①
            }
        });
    }
    // 重写 onKeyDown 方法，该方法可监听它所包含的所有组件的按键被按下事件
    @Override
```

```
    public boolean onKeyDown(int keyCode, KeyEvent event)
    {
        super.onKeyDown(keyCode , event);
        Log.v("-Activity-" , "the onKeyDown in Activity");
        //返回 false，表明并未完全处理该事件，该事件依然向外扩散
        return false;
    }
}
```

从上面的程序可以看出，粗体字代码重写了 Activity 的 onKeyDown(int keyCode, KeyEvent event)方法，当该 Activity 包含的所有组件上按下某个键时，该方法都可能被触发——只要该组件没有完全处理该事件。

不仅如此，上面的程序还采用监听模式来处理该按钮上的按键被按下的事件。

运行上面的程序，先把焦点移动到程序界面的按钮上，然后按下模拟器右边的按键，将可以在 DDMS 的 LogCat 中看到如图 3.7 所示的输出。



图 3.7　基于回调的事件传播

从图 3.7 不难看出，当该组件上发生某个按键被按下的事件时，Android 系统最先触发的应该是该按键上绑定的事件监听器，接着才触发该组件提供的事件回调方法，然后还会传播到该组件所在的 Activity——但如果我们让任何一个事件处理方法返回了 true，那么该事件将不会继续向外传播。例如我们改写上面的 Activity 代码，将程序中①号代码改为 return true，然后再运行该程序、把焦点定位到该按钮上之后单击某个按键，在 DDMS 的 LogCat 中将看到如图 3.8 所示的输出。



图 3.8　监听器阻止事件传播

3.3.3　重写 onTouchEvent 方法响应触摸屏事件

对比 Android 提供的两种事件处理模型，不难发现基于监听的事件处理模型具有更大的优势：

- 基于监听的事件模型分工更明确，事件源、事件监听由两个类分开实现，因此具有更好的可维护性。
- Android 的事件处理机制保证基于监听的事件监听器会被优先触发。

实例：通过回调实现跟随手指的小球

在某些特定情况下，基于回调的事件处理机制会更好地提高程序的内聚性，例如上一章

的实例：跟随手指的小球，如果改为基于回调的实现，可以更好地提高程序的内聚性。

例如将该实例中的 DrawView 类改为如下形式。

程序清单：codes\03\3.3\CustomView\src\org\crazyit\event\DrawView.java

```
public class DrawView extends View
{
    public float currentX = 40;
    public float currentY = 50;
    // 定义、创建画笔
    Paint p = new Paint();
    public DrawView(Context context, AttributeSet set)
    {
        super(context, set);
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 设置画笔的颜色
        p.setColor(Color.RED);
        // 绘制一个小圆（作为小球）
        canvas.drawCircle(currentX, currentY, 15, p);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 当前组件的 currentX、currentY 两个属性
        this.currentX = event.getX();
        this.currentY = event.getY();
        // 通知改组件重绘
        this.invalidate();
        // 返回 true 表明处理方法已经处理该事件
        return true;
    }
}
```

上面的程序中粗体字代码重写了 View 组件的 onTouchEvent(MotionEvent event)方法，这表示该组件自己就可处理触摸屏事件，当用户手指在屏幕上移动时，该 View 上绘制的小球将会跟随用户手指。也就是说，这个自定义 View 本身就可以很好地处理屏幕事件。

接下来在程序中使用这个 DrawView 几乎不需增加任何处理，直接将 View 放到界面布局中即可。如下面界面布局的 XML 文件所示。

程序清单：codes\03\3.3\CustomView\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 使用自定义组件 -->
<org.crazyit.event.DrawView
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"/>
</LinearLayout>
```

接下来 Activity 类中不需要为该 View 绑定事件监听器——因为该 View 自己就可以处理

它的触摸屏事件。

通过为View提供事件处理的回调方法，可以很好地把事件处理方法封装在该View内部，从而提高程序的内聚性——基于回调的事件处理更适合于应付那种事件处理逻辑比较固定的View，比如上面介绍的这个跟随用户手指的View。

3.4 响应的系统设置的事件

在开发Android应用时，有时候可能需要让应用程序随系统设置而进行调整，比如判断系统的屏幕方向、判断系统方向的方向导航设备等。除此之外，有时候可能还需要让应用程序监听系统设置的更改，对系统设置的更改做出响应。

3.4.1 Configuration类简介

Configuration类专门用于描述手机设备上的配置信息，这些配置信息既包括用户特定的配置项，也包括系统的动态设备配置。

程序可调用Activity的如下方法来获取系统的Configuration对象：

```
Configuration cfg = getResources().getConfiguration();
```

一旦获得了系统的Configuration对象，该对象提供了如下常用属性来获取系统的配置信息。

- public float fontScale：获取当前用户设置的字体的缩放因子。
- public int keyboard：获取当前设备所关联的键盘类型。该属性可能返回如下值：KEYBOARD_NOKEYS、KEYBOARD_QWERTY（普通电脑键盘）、KEYBOARD_12KEY（只有12个键的小键盘）。
- public int keyboardHidden：该属性返回一个boolean值用于标识当前键盘是否可用。该属性不仅会判断系统的硬件键盘，也会判断系统的软键盘（位于屏幕上）。如果该系统的硬件键盘不可用，但软键盘可用，该属性也会返回KEYBOARDHIDDEN_NO；只有当两个键盘都不可用时才返回KEYBOARDHIDDEN_YES。
- public Locale locale：获取用户当前的Locale。
- public int mcc：获取移动信号的国家码。
- public int mnc：获取移动信号的网络码。
- public int navigation：判断系统上方向导航设备的类型。该属性可能返回如NAVIGATION_NONAV（无导航）、NAVIGATION_DPAD（DPAD导航）、NAVIGATION_TRACKBALL（轨迹球导航）、NAVIGATION_WHEEL（滚轮导航）等属性值。
- public int orientation：获取系统屏幕的方向，该属性可能返回ORIENTATION_LANDSCAPE（横向屏幕）、ORIENTATION_PORTRAIT（竖向屏幕）、ORIENTATION_SQUARE（方形屏幕）等属性值。
- public int touchscreen：获取系统触摸屏的触摸方式。该属性可能返回TOUCHSCREEN_NOTOUCH（无触摸屏）、TOUCHSCREEN_STYLUS（触摸笔式的触摸

屏）、TOUCHSCREEN_FINGER（接受手指的触摸屏）。

下面以一个实例来介绍Configuration的用法，该程序可以获取系统的屏幕方向、触摸屏方式等。

实例：获取系统设备状态

该程序的界面布局比较简单，程序只是提供了4个文本框来显示系统的屏幕方向、触摸屏方式等状态，故此处不再给出界面布局文件。

该程序的Java代码主要可分为两步：

① 获取系统的Configuration对象。

② 调用Configuration对象的属性来获取设备状态。

下面是该程序的Java代码。

程序清单：codes\03\3.4\ConfigurationTest\src\org\crazyit\cfg\ConfigurationTest.java

```
public class ConfigurationTest extends Activity
{
    EditText ori;
    EditText navigation;
    EditText touch;
    EditText mnc;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取应用界面中的界面组件
        ori = (EditText)findViewById(R.id.ori);
        navigation = (EditText)findViewById(R.id.navigation);
        touch = (EditText)findViewById(R.id.touch);
        mnc = (EditText)findViewById(R.id.mnc);
        Button bn = (Button)findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            // 为按钮绑定事件监听器
            @Override
            public void onClick(View source)
            {
                // 获取系统的Configuration对象
                Configuration cfg = getResources().getConfiguration();
                String screen = cfg.orientation ==
                    Configuration.ORIENTATION_LANDSCAPE
                    ? "横向屏幕": "竖向屏幕";
                String mncCode = cfg.mnc + "";
                String naviName = cfg.orientation ==
                    Configuration.NAVIGATION_NONAV
                    ? "没有方向控制" :
                    cfg.orientation == Configuration.NAVIGATION_WHEEL
                    ? "滚轮控制方向" :
                    cfg.orientation == Configuration.NAVIGATION_DPAD
                    ? "方向键控制方向" : "轨迹球控制方向";
                navigation.setText(naviName);
                String touchName = cfg.touchscreen ==
                    Configuration.TOUCHSCREEN_NOTOUCH
                    ? "无触摸屏" : "支持触摸屏";
                ori.setText(screen);
                mnc.setText(mncCode);
```

```
                touch.setText(touchName);
            }
        });
    }
}
```

上面的程序中粗体字代码用于获取系统的 Configuration 对象，一旦获得了系统的 Configuration 之后，程序就可以通过它来了解系统的设备状态了。运行上面的程序，将显示如图 3.9 所示界面。

单击图 3.9 所示界面中“获取手机信息”按钮，系统显示如图 3.10 所示界面。



图 3.9　未获取设备状态之前



图 3.10　获取设备状态

3.4.2　重写 onConfigurationChanged 响应系统设置更改

如果程序需要监听系统设置的更改，则可以考虑重写 Activity 的 onConfigurationChanged (Configuration newConfig)方法，该方法是一个基于回调的事件处理方法：当系统设置发生更改时，该方法会被自动触发。

为了在程序中动态地更改系统设置，我们可调用 Activity 的 setRequestedOrientation(int)方法来修改屏幕的方向。

实例：监听屏幕方向改变

该实例的界面布局很简单，该界面中仅包含一个普通按钮，该按钮用于动态修改系统屏幕的方向，此处不再给出系统界面布局代码。

该程序的 Java 代码主要会调用 Activity 的 setRequestedOrientation(int)方法来动态更改屏幕方向。除此之外，我们还重写 Activity 的 onConfigurationChanged(Configuration newConfig)方法，该方法可用于监听系统设置的更改。程序代码如下。

程序清单：codes\03\3.4\ChangeCfg\src\org\crazyit\cfg\ChangeCfg.java

```
public class ChangeCfg extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Configuration config = getResources().getConfiguration();
```

```
                        // 如果当前是横屏
                        if (config.orientation == Configuration.ORIENTATION_LANDSCAPE)
                        {
                            // 设为竖屏
                            ChangeCfg.this.setRequestedOrientation(
                                ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
                        }
                        // 如果当前是竖屏
                        if (config.orientation == Configuration.ORIENTATION_PORTRAIT)
                        {
                            // 设为横屏
                            ChangeCfg.this.setRequestedOrientation(
                                ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
                        }
                    }
                });
        }
        // 重写该方法，用于监听系统设置的更改，主要是监控屏幕方向的更改
        @Override
        public void onConfigurationChanged(Configuration newConfig)
        {
            super.onConfigurationChanged(newConfig);
            String screen = newConfig.orientation ==
                Configuration.ORIENTATION_LANDSCAPE ? "横向屏幕" : "竖向屏幕";
            Toast.makeText(this, "系统的屏幕方向发生改变" + "\n修改后的屏幕方向为："
                + screen, Toast.LENGTH_LONG).show();
        }
    }
```

上面的程序中前两行粗体字代码用于动态地修改手机屏幕的方向，接下来的粗体字代码重写了 Activity 的 onConfigurationChanged(Configuration newConfig)方法，当系统设置发生更改时，该方法将会被自动回调。

除此之外，为了让该 Activity 能监听屏幕方向更改的事件，需要在配置该 Activity 时指定 android:configChanges 属性，该属性可以支持 mcc、mnc、locale、touchscreen、keyboard、keyboardHidden、navigation、orientation、screenLayout、uiMode、screenSize、smallestScreenSize、fontScale 属性值，其中 orientation 属性值指定该 Activity 可以监听屏幕方向改变的事件。

因此将应用的 AndroidManifest.xml 文件改为如下形式。

程序清单：codes\03\3.4\ChangeCfg\AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.crazyit.cfg"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="12" />
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">
        <!-- 设置Activity可以监听屏幕方向改变的事件 -->
        <activity android:configChanges="orientation"
            android:name=".ChangeCfg"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
```

```
        </intent-filter>
                </activity>
            </application>
        </manifest>
```



图 3.11　设置横向屏幕并响应系统设置的更改

上面的粗体字配置代码指定了该 Activity 可以监听屏幕方向改变的事件，这样当程序改变手机屏幕方向时，Activity 的 onConfigurationChanged()方法就会被回调。

提供上面的程序和设置之后，运行该程序，单击应用程序中“更改屏幕方向”按钮，将可以看到如图 3.11 所示界面。

注意：

该程序的<uses-sdk.../>元素的 android:targetSdkVersion 属性最高只能设置成 12，如果该属性设置得过高，那么 onConfigurationChanged()方法不会被激发。



3.5　Handler 消息传递机制

出于性能优化考虑，Android 的 UI 操作并不是线程安全的，这意味着如果有多个线程并发操作 UI 组件，可能导致线程安全问题。为了解决这个问题，Android 制定了一条简单的规则：只允许 UI 线程修改 Activity 里的 UI 组件。

当一个程序第一次启动时，Android 会同时启动一条主线程（Main Thread），主线程主要负责处理与 UI 相关的事件，如用户的按键事件、用户接触屏幕的事件及屏幕绘图事件，并把相关的事件分发到对应的组件进行处理。所以主线程通常又被叫做 UI 线程。

Android 的消息传递机制是另一种形式的“事件处理”，这种机制主要是为了解决 Android 应用的多线程问题——Android 平台只允许 UI 线程修改 Activity 里的 UI 组件，这样就会导致新启动的线程无法动态改变界面组件的属性值。但在实际 Android 应用开发中，尤其是涉及动画的游戏开发中，需要让新启动的线程周期性地改变界面组件的属性值，这就需要借助于 Handler 的消息传递机制来实现了。

3.5.1　Handler 类简介

Handler 类的主要作用有两个：

- 在新启动的线程中发送消息。
- 在主线程中获取、处理消息。

上面的说法看上去很简单，似乎只要分成两步即可：在新启动的线程中发送消息；然后在主线程中获取、并处理消息。但这个过程涉及一个问题：新启动的线程何时发送消息呢？主线程何时去获取并处理消息呢？这个时机显然不好控制。

为了让主线程能“适时”地处理新启动的线程所发送的消息，显然只能通过回调的方式来实现——开发者只要重写 Handler 类中处理消息的方法，当新启动的线程发送消息时，消息会发送到与之关联的 MessageQueue，而 Handler 会不断地从 MessageQueue 中获取并处理消息——这将导致 Handler 类中处理消息的方法被回调。

Handler 类包含如下方法用于发送、处理消息。

- **void handleMessage(Message msg)**：处理消息的方法。该方法通常用于被重写。

- final boolean hasMessages(int what)：检查消息队列中是否包含what属性为指定值的消息。
- final boolean hasMessages(int what, Object object)：检查消息队列中是否包含what属性为指定值且object属性为指定对象的消息。
- 多个重载的Message obtainMessage()：获取消息。
- sendEmptyMessage(int what)：发送空消息。
- final boolean sendEmptyMessageDelayed(int what, long delayMillis)：指定多少毫秒之后发送空消息。
- final boolean sendMessage(Message msg)：立即发送消息。
- final boolean sendMessageDelayed(Message msg, long delayMillis)：指定多少毫秒之后发送消息。

借助于上面这些方法，程序可以方便地利用Handler来进行消息传递。

实例：自动播放动画

下面的程序通过一个新线程来周期性地修改ImageView所显示的图片，通过这种方式来开发一个动画效果。该程序的界面布局代码非常简单，程序只是在界面布局中定义了ImageView组件，此处不再给出界面布局代码。

接下来主程序使用java.util.Timer来周期性地执行指定任务，程序代码如下。

程序清单：codes\03\3.5\HandlerTest\src\org\crazyit\handler\HandlerTest.java

```
public class HandlerTest extends Activity
{
    // 定义周期性显示的图片的ID
    int[] imageIds = new int[]
    {
        R.drawable.java,
        R.drawable.ee,
        R.drawable.ajax,
        R.drawable.xml,
        R.drawable.classic
    };
    int currentImageId = 0;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ImageView show = (ImageView) findViewById(R.id.show);
        final Handler myHandler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 如果该消息是本程序所发送的
                if (msg.what == 0x1233)
                {
                    // 动态地修改所显示的图片
                    show.setImageResource(imageIds[currentImageId++
                        % imageIds.length]);
                }
            }
        };
```

```
        // 定义一个计时器，让该计时器周期性地执行指定任务
        new Timer().schedule(new TimerTask()
        {
            @Override
            public void run()
            {
                // 发送空消息
                myHandler.sendEmptyMessage(0x1233);
            }
        }, 0, 1200);
    }
}
```

上面的程序中下面的粗体字代码通过 Timer 周期性地执行指定任务，Timer 对象可调度 TimerTask 对象，TimerTask 对象的本质就是启动一条新线程，由于 Android 不允许在新线程中访问 Activity 里的界面组件，因此程序只能在新线程里发送一条消息，通知系统更新 ImageView 组件。

上面的程序的第一段粗体字代码重写了 Handler 的 handleMessage(Message msg)方法，该方法用于处理消息——当新线程发送消息时，该方法会被自动回调，handleMessage(Message msg)方法依然位于主线程，所以可以动态地修改 ImageView 组件的属性。这就实现了本程序所要达到的效果：由新线程来周期性地修改 ImageView 的属性，从而实现动画效果。运行上面的程序可看到应用程序中 5 张图片交替显示的动态效果。

3.5.2 Handler、Loop、MessageQueue 的工作原理

为了更好地理解 Handler 的工作原理，先介绍一下与 Handler 一起工作的几个组件。

- Message：Handler 接收和处理的消息对象。
- Looper：每个线程只能拥有一个 Looper。它的 loop 方法负责读取 MessageQueue 中的消息，读到信息之后就把消息交给发送该消息的 Handler 进行处理。
- MessageQueue：消息队列，它采用先进先出的方式来管理 Message。程序创建 Looper 对象时会在它的构造器中创建 Looper 对象。Looper 提供的构造器源代码如下：

```
private Looper()
{
    mQueue = new MessageQueue();
    mRun = true;
    mThread = Thread.currentThread();
}
```

该构造器使用了 private 修饰，表明程序员无法通过构造器创建 Looper 对象。从上面的代码不难看出，程序在初始化 Looper 时会创建一个与之关联的 MessageQueue，这个 MessageQueue 就负责管理消息。

- Handler：它的作用有两个——发送消息和处理消息，程序使用 Handler 发送消息，被 Handler 发送的消息必须被送到指定的 MessageQueue。也就是说，如果希望 Handler 正常工作，必须在当前线程中有一个 MessageQueue，否则消息就没有 MessageQueue 进行保存了。不过 MessageQueue 是由 Looper 负责管理的，也就是说，如果希望 Handler 正常工作，必须在当前线程中有一个 Looper 对象。为了保证当前线程中有 Looper 对象，可以分如下两种情况处理。

- 主UI线程中，系统已经初始化了一个Looper对象，因此程序直接创建Handler即可，然后就可通过Handler来发送消息、处理消息。
- 程序员自己启动的子线程，程序员必须自己创建一个Looper对象，并启动它。创建Looper对象调用它的prepare()方法即可。

prepare()方法保证每个线程最多只有一个Looper对象。prepare()方法的源代码如下：

```
public static final void prepare()
{
      if (sThreadLocal.get() != null)
      {
            throw new RuntimeException("Only one Looper may be created per thread");
      }
      sThreadLocal.set(new Looper());
}
```

然后调用Looper的静态loop()方法来启动它。loop()方法使用一个死循环不断取出MessageQueue中的消息，并将取出的消息分给该消息对应的Handler进行处理。下面是Looper类的的loop()方法的源代码：

```
for (;;)
{
      Message msg = queue.next(); // 获取消息队列的下一个消息，如果没有消息，将会阻塞
      if (msg == null) {
            // 如果消息为null，表明消息队列正在退出
            return;
      }
      Printer logging = me.mLogging;
      if (logging != null) {
            logging.println(">>>>> Dispatching to " + msg.target + " " +
                  msg.callback + ": " + msg.what);
      }
      msg.target.dispatchMessage(msg);
      if (logging != null) {
            logging.println("<<<<< Finished to " + msg.target + " " + msg.callback);
      }
      // 使用final修饰该标识符，保证在分发消息的过程中线程标识符不会被修改
      final long newIdent = Binder.clearCallingIdentity();
      if (ident != newIdent) {
      Log.wtf(TAG, "Thread identity changed from 0x"
            + Long.toHexString(ident) + " to 0x"
            + Long.toHexString(newIdent) + " while dispatching to "
            + msg.target.getClass().getName() + " "
            + msg.callback + " what=" + msg.what);
      }
      msg.recycle();
}
```

归纳起来，Looper、MessageQueue、Handler各自的作用如下。

- Looper：每个线程只有一个Looper，它负责管理MessageQueue，会不断地从MessageQueue中取出消息，并将消息分给对应的Handler处理。
- MessageQueue：由Looper负责管理。它采用先进先出的方式来管理Message。
- Handler：它能把消息发送给Looper管理的MessageQueue，并负责处理Looper分给它的消息。

在线程中使用Handler的步骤如下。

1. 调用Looper的prepare()方法为当前线程创建Looper对象，创建Looper对象时，它

的构造器会创建与之配套的 MessageQueue。

② 有了 Looper 之后，创建 Handler 子类的实例，重写 handleMessage()方法，该方法负责处理来自于其他线程的消息。

③ 调用 Looper 的 loop()方法启动 Looper。

下面通过一个实例来介绍 Looper 与 Handler 的用法。

实例：使用新线程计算质数

该实例允许用户输入一个数值上限，当用户单击“计算”按钮时，该应用会将该上限数值发送到新启动的线程，让该线程来计算该范围内的所有质数。

之所以不直接在 UI 线程中计算该范围的所有质数，是因为 UI 线程需要响应用户动作，如果在 UI 线程中执行一个“耗时”操作，将会导致 UI 线程被阻塞，从而让应用程序失去响应。比如在该实例中，如果用户输入的数值太大，系统可能需要较长时间才能计算出所有质数，这就可能导致 UI 线程失去响应。



提示：

尽量避免在 UI 线程中执行耗时操作，因为这样可能导致一个“著名”的异常：ANR 异常。只要在 UI 线程中执行需要消耗大量时间的操作，都会引发 ANR，因为这会导致 Android 应用程序无法响应输入事件和 Broadcast。

为了将用户在 UI 界面输入的数值上限动态地传给新启动的线程，本实例将会在线程中创建一个 Handler 对象，然后 UI 线程的事件处理方法就可以通过该 Handler 向新线程发送消息了。

该实例的界面布局文件比较简单，只有一个文本框和一个按钮。故此处不再给出界面布局文件。

该实例的 Activity 代码如下。

程序清单：codes\03\3.5\CalPrime\src\org\crazyit\handler\CalPrime.java

```
public class CalPrime extends Activity
{
    static final String UPPER_NUM = "upper";
    EditText etNum;
    CalThread calThread;
    // 定义一个线程类
    class CalThread extends Thread
    {
        public Handler mHandler;
        public void run()
        {
            Looper.prepare();
            mHandler = new Handler()
            {
                // 定义处理消息的方法
                @Override
                public void handleMessage(Message msg)
                {
                    if(msg.what == 0x123)
                    {
                        int upper = msg.getData().getInt(UPPER_NUM);
                        List<Integer> nums = new ArrayList<Integer>();
                        // 计算从 2 开始、到 upper 的所有质数
                        outer:
```

```
                                for (int i = 2 ; i <= upper ; i++)
                                {
                                    // 用i除以从2开始、到i的平方根的所有数
                                    for (int j = 2 ; j <= Math.sqrt(i) ; j++)
                                    {
                                        // 如果可以整除，则表明这个数不是质数
                                        if(i != 2 && i % j == 0)
                                        {
                                            continue outer;
                                        }
                                    }
                                    nums.add(i);
                                }
                                // 使用Toast显示统计出来的所有质数
                                Toast.makeText(CalPrime.this , nums.toString()
                                    , Toast.LENGTH_LONG).show();
                            }
                        }
                    };
                    Looper.loop();
                }
            }
            @Override
            public void onCreate(Bundle savedInstanceState)
            {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                etNum = (EditText)findViewById(R.id.etNum);
                calThread = new CalThread();
                // 启动新线程
                calThread.start();
            }
            // 为按钮的点击事件提供事件处理函数
            public void cal(View source)
            {
                // 创建消息
                Message msg = new Message();
                msg.what = 0x123;
                Bundle bundle = new Bundle();
                bundle.putInt(UPPER_NUM ,
                    Integer.parseInt(etNum.getText().toString()));
                msg.setData(bundle);
                // 向新线程中的Handler发送消息
                calThread.mHandler.sendMessage(msg);
            }
        }
```

上面的粗体字代码是实例的关键代码，这些粗体字代码在新线程内创建了一个Handler，由于在新线程中创建Handler时必须先创建Looper，因此程序先调用Looper的prepare()方法为当前线程创建了一个Looper实例，并创建配套的MessageQueue，新线程有了Looper对象之后，接下来程序创建了一个Handler对象，该Handler可以处理其他线程发送过来的消息。程序最后还调用了Looper的loop()方法。

运行该程序，无论用户输入多大的数值，计算该范围的质数将会交给新线程完成，而前台UI线程不会受到影响。该程序运行效果如图3.12所示。

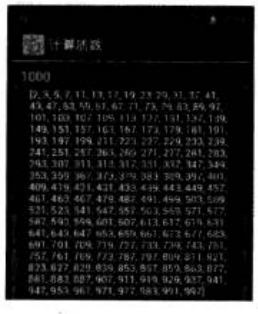


图3.12 使用新线程计算质数

3.6 异步任务（AsyncTask）

前面已经介绍过，Android 的 UI 线程主要负责处理用户的按键事件、用户触屏事件及屏幕绘图事件等，因此开发者的其他操作不应该、也不能阻塞 UI 线程，否则 UI 界面将会变得停止响应——用户感觉非常糟糕。



提示：

Android 默认约定当 UI 线程阻塞超过 20 秒将会引发 ANR（Application Not Responding）异常。但实际上，不要说 20 秒，即使是 5 秒甚至 2 秒，用户都会感觉十分不爽（用户是很急燥的！）。因此笔者认为，其实没必要去记这个 20 秒的时间限度。总之，开发者需要牢记：不要在 UI 线程中执行一些耗时的操作。

为了避免 UI 线程失去响应的问题，Android 建议将耗时操作放在新线程中完成，但新线程也可能需要动态更新 UI 组件：比如需要从网上获取一个网页，然后在 TextView 中将其源代码显示出来，此时就应该将连接网络、获取网络数据的操作放在新线程中完成。问题是：获取网络数据之后，新线程不允许直接更新 UI 组件。

为了解决新线程不能更新 UI 组件的问题，Android 提供了如下几种解决方案。

- 使用 Hanlder 实现线程之间的通信。
- Activity.runOnUiThread(Runnable)。
- View.post(Runnable)。
- View.postDelayed(Runnable, long)。

上一节已经见到了使用 Handler 的示例，后面的三种方式可能导致编程略显烦琐，而异步任务（AsyncTask）则可进一步简化这种操作。

AsyncTask<>是一个抽象类，通常用于被继承，继承 AsyncTask 时需要指定如下三个泛型参数。

相对来说 AsyncTask 更轻量级一些，适用于简单的异步处理，不需要借助线程和 Handler 即可实现。

AsyncTask<Params, Progress, Result>是抽象类，它定义了如下三种泛型类型。

- Params：启动任务执行的输入参数的类型。
- Progress：后台任务完成的进度值的类型。
- Result：后台执行任务完成后返回结果的类型。

使用 AsyncTask 只要如下三步即可。

① 创建 AsyncTask 的子类，并为三个泛型参数指定类型。如果某个泛型参数不需要指定类型，可将它指定为 Void。

② 根据需要，实现 AsyncTask 的如下方法。

- doInBackground(Params...)：重写该方法就是后台线程将要完成的任务。该方法可以调用 publishProgress(Progress... values)方法更新任务的执行进度。
- onProgressUpdate(Progress... values)：在 doInBackground()方法中调用 publishProgress()方法更新任务的执行进度后，将会触发该方法。
- onPreExecute()：该方法将在执行后台耗时操作前被调用。通常该方法用于完成一些初始化的准备工作，比如在界面上显示进度条等。

- onPostExecute(Result result)：当doInBackground()完成后，系统会自动调用onPostExecute()方法，并将doInBackground()方法的返回值传给该方法。

3 调用AsyncTask子类的实例的execute(Params... params)开始执行耗时任务。

使用AsyncTask时必须遵守如下规则。

- 必须在UI线程中创建AsyncTask的实例。
- 必须在UI线程中调用AsyncTask的execute()方法。
- AsyncTask的onPreExecute()、onPostExecute(Result result)、doInBackground(Params... params), onProgressUpdate(Progress... values)方法，不应该由程序员代码调用，而是由Android系统负责调用。
- 每个AsyncTask只能被执行一次，多次调用将会引发异常。

实例：使用异步任务下载

下面的实例示范如何使用异步任务下载网络资源。该实例的界面布局很简单，只包含两个组件：一个文本框用于显示从网络下载的页面代码；一个按钮用于激发下载任务。由于该实例的界面布局很简单，故此处不再给出界面布局文件。

该程序的Activity代码如下：

程序清单：coder\03\3.6\AsyncTaskTest\src\org\crazyit\handler\AsyncTaskTest.java

```
public class AsyncTaskTest extends Activity
{
    private TextView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (TextView) findViewById(R.id.show);
    }
    // 重写该方法，为界面的按钮提供事件响应方法
    public void download(View source) throws MalformedURLException
    {
        DownTask task = new DownTask(this);
        task.execute(new URL("http://www.crazyit.org/ethos.php"));
    }
    class DownTask extends AsyncTask<URL, Integer, String>
    {
        // 可变长的输入参数，与AsyncTask.exucute()对应
        ProgressDialog pdialog;
        // 定义记录已经读取行的数量
        int hasRead = 0;
        Context mContext;
        public DownTask(Context ctx)
        {
            mContext = ctx;
        }
        @Override
        protected String doInBackground(URL... params)
        {
            StringBuilder sb = new StringBuilder();
            try
            {
                URLConnection conn = params[0].openConnection();
                // 打开conn连接对应的输入流，并将它包装成BufferedReader
```

```
                BufferedReader br = new BufferedReader(
                        new InputStreamReader(conn.getInputStream()
                        , "utf-8"));
                String line = null;
                while ((line = br.readLine()) != null)
                {
                        sb.append(line + "\n");
                        hasRead++;
                        publishProgress(hasRead);
                }
                return sb.toString();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            return null;
        }
        @Override
        protected void onPostExecute(String result)
        {
            // 返回 HTML 页面的内容
            show.setText(result);
            pdialog.dismiss();
        }
        @Override
        protected void onPreExecute()
        {
            pdialog = new ProgressDialog(mContext);
            // 设置对话框的标题
            pdialog.setTitle("任务正在执行中");
            // 设置对话框显示的内容
            pdialog.setMessage("任务正在执行中，敬请等待...");
            // 设置对话框不能用“取消”按钮关闭
            pdialog.setCancelable(false);
            // 设置该进度条的最大进度值
            pdialog.setMax(202);
            // 设置对话框的进度条风格
            pdialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            // 设置对话框的进度条是否显示进度
            pdialog.setIndeterminate(false);
            pdialog.show();
        }
        @Override
        protected void onProgressUpdate(Integer... values)
        {
            // 更新进度
            show.setText("已经读取了【" + values[0] + "】行！");
            pdialog.setProgress(values[0]);
        }
    }
}
```

上面程序的 download()方法很简单，它只是创建了 DownTask（AsyncTask 的子类）实例，并调用它的 execute()方法开始执行异步任务。

该程序的重点是实现 AsyncTask 的子类，实现该子类时实现了如下 4 个方法。

- doInBackground()：该方法的代码完成实际的下载任务。
- onPreExecute()：该方法的代码负责在下载开始的时候显示一个进度条。
- onProgressUpdate()：该方法的代码负责随着下载进度的改变更新进度条的进度值。

- onPostExecute()：该方法的代码负责当下载完成后、将下载的代码显示出来。



提示：

该程序使用了网络编程从网络下载数据。关于 Android 网络编程的知识，请参考本书第 13 章的内容。除此之外，本程序需要访问网络，因此还需要在 AndroidManifest.xml 文件中声明如下权限：<uses-permission android:name="android.permission.INTERNET"/>

运行该程序并单击“下载”按钮，将可以看到如图 3.13 所示界面。



图 3.13 使用异步任务下载网络资源

3.7 本章小结

本章是对上一章内容的补充：图形界面编程肯定需要与事件处理结合，当我们开发了一个界面友好的应用之后，用户在程序界面上执行操作时，程序必须为这种用户操作提供响应动作，这种响应动作就是由事件处理来完成的。学习本章的重点是掌握 Android 的两种事件处理机制：基于回调的事件处理和基于监听的事件处理；对于基于监听的事件处理来说，开发者需要掌握事件监听的处理模式，以及不同事件对应的监听器接口；对于基于回调的事件处理来说，开发者需要掌握不同事件对应的回调方法。除此之外，本章还介绍了重写 onConfigurationChanged 方法响应系统设置更改。需要指出的是，由于 Android 不允许在子线程中更新界面组件，如果想在子线程中更新界面组件，开发者需要借助于 Handler 对象来实现。本章详细介绍了 Handler、Looper 与 MessageQueue 之间的关系及工作原理。

CHAPTER

4

第4章 深入理解 Activity 与 Fragment

本章要点

- 理解 Activity 的功能与作用
- 开发普通 Activity 类
- 在 AndroidManifest.xml 中配置 Activity
- 特殊 Activity 的功能和用法
- 在程序中启动 Activity
- 关闭 Activity
- 使用 Bundle 在不同 Activity 之间交换数据
- 启动其他 Activity 并返回结果
- Activity 的回调机制
- Activity 的生命周期
- Fragment 概述及 Fragment 的设计哲学
- 开发 Fragment
- 使用 ListFragment
- 管理 Fragmeng、并让 Fragment 与 Activity 通信
- Fragment 的生命周期

Activity 是 Android 应用的重要组成单元之一（另外三个是 Service、BroadcastReceiver 和 ContentProvider），而 Activity 又是 Android 应用最常见的组件之一。前面看到的示例通常都只包含一个 Activity，但在实际应用中这是不大可能的，实际应用中往往包括多个 Activity，不同的 Activity 向用户呈现不同的操作界面。Android 应用的多个 Activity 组成 Activity 栈，当前活动的 Activity 位于栈顶。

有 Web 开发经验的读者对 Servlet 的概念应该比较熟悉了。实际上 Activity 对于 Android 应用的作用有点类似于 Servlet 对于 Web 应用的作用——一个 Web 应用通常都需要 N 个 Servlet 组成（JSP 的本质依然是 Servlet）；那么一个 Android 应用通常也需要 N 个 Activity 组成。对于 Web 应用而言，Servlet（把 JSP 也统一成 Servlet）主要负责与用户交互，并向用户呈现应用状态；对于 Android 应用而言，Activity 大致也具有相同的功能。

当 Activity 处于 Android 应用运行时，同样受系统控制有其自身的生命周期，本章深入介绍 Activity 的相关知识。

4.1 建立、配置和使用 Activity

Activity 是 Android 应用中最重要、最常见的应用组件（此处的组件是粗粒度的系统组成部分，并非指界面控件：widget）。Android 应用开发的一个重要组成部分就是开发 Activity，下面将会详细介绍 Activity 开发、配置的相关知识。

4.1.1 Activity

与开发 Web 应用时建立 Servlet 类相似，建立自己的 Activity 也需要继承 Activity 基类。当然，在不同应用场景下，有时也要求继承 Activity 的子类。例如如果应用程序界面只包括列表，则可以让应用程序继承 ListActivity；如果应用程序界面需要实现标签页效果，则可以让应用程序继承 TabActivity。

图 4.1 显示了 Android 提供的 Activity 类。

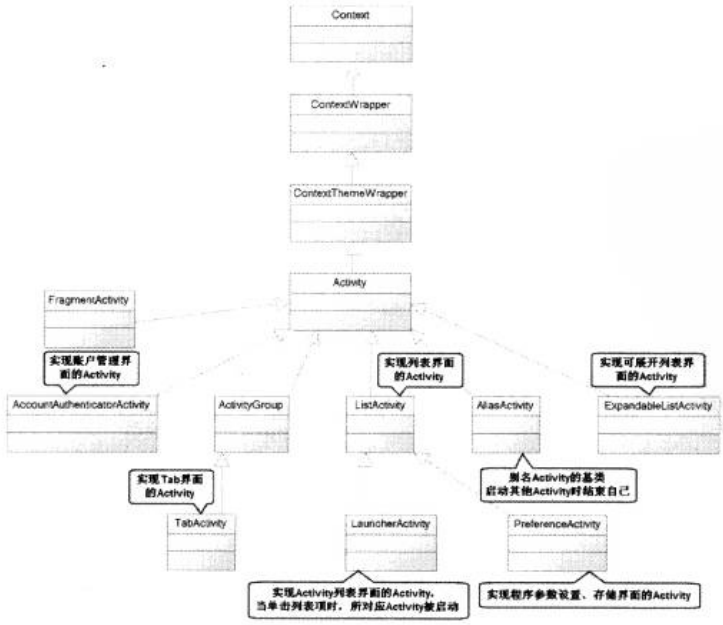


图 4.1 各种 Activity 基类

如图 4.1 所示，Activity 类间接或直接地继承了 Context、ContextWrapper、ContextTheme Wrapper 等基类，因此 Activity 可以直接调用它们的方法。

与 Servlet 类似，当一个 Activity 类定义出来之后，这个 Activity 类何时被实例化、它所包含的方法何时被调用，这些都不是由开发者决定的，都应该由 Android 系统来决定。

为了让 Servlet 能响应用户请求，开发者需要重写 HttpServlet 的 doRequest(..)、doResponse(..)方法，或重写 service(..)方法。Activity 与此类似，创建一个 Activity 也需要实现一个或多个方法，其中最常见的就是实现 onCreate(Bundle status)方法，该方法将会在 Activity 创建时被回调，该方法调用 Activity 的 setContentView(View view)方法来显示要展示的 View。为了管理应用程序界面中的各组件，调用 Activity 的 findViewById(int id)方法来获取程序界面中的组件，接下来去修改各组件的属性和方法即可。

实例：用 LauncherActivity 开发启动 Activity 的列表

通过前几章的实例已经介绍了 Activity、ListActivity、TabActivity 等基类的用法，接下来开发一个继承 LauncherActivity 的应用。

LauncherActivity 继承了 ListActivity，因此它本质上也是一个开发列表界面的 Activity，但它开发出来的列表界面与普通列表界面有所不同。它开发出来的列表界面中的每个列表项都对应于一个 Intent，因此当用户单击不同的列表项时，应用程序会自动启动对应的 Activity。

使用 LauncherActivity 的方法并不难，由于依然是一个 ListActivity，因此同样需要为它设置 Adapter——既可使用简单的 ArrayAdapter，也可使用 SimpleAdapter，当然也可以扩展 BaseAdapter 来实现自己的 Adapter。与使用普通 ListActivity 不同的是，继承 LauncherActivity 时通常应该重写 Intent intentForPosition(int position)方法，该方法根据不同列表项返回不同的 Intent（用于启动不同的 Activity）。

下面的程序是一个 LauncherActivity 的子类。

程序清单：codes\04\4.1\OtherActivity\src\org\crazyit\app\OtherActivity.java

```
public class OtherActivity extends LauncherActivity
{
    //定义两个 Activity 的名称
    String[] names = {"设置程序参数" , "查看星际兵种"};
    //定义两个 Activity 对应的实现类
    Class<?>[] clazzs = {PreferenceActivityTest.class
        , ExpandableListActivityTest.class};
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_list_item_1 , names);
        // 设置该窗口显示的列表所需的 Adapter
        setListAdapter(adapter);
    }
    //根据列表项返回指定 Activity 对应的 Intent
    @Override public Intent intentForPosition(int position)
    {
        return new Intent(OtherActivity.this , clazzs[position]);
    }
}
```

上面的程序中第一行粗体字代码为该 ListActivity 设置了所需的内容 Adapter，第二段粗

体字代码则根据用户单击的列表项去启动对应的 Activity。

上面的程序还用到了如下两个 Activity。

- ExpandableListActivityTest：它是 ExpandableListActivity 的子类，用于显示一个可展开的列表窗口。
- PreferenceActivityTest：它是 PreferenceActivity 的子类，用于显示一个显示设置选项参数并进行保存的窗口。

实例：使用 ExpandableListActivity 实现可展开的 Activity

先看 ExpandableListActivityTest，它继承了 ExpandableListActivity 基类，ExpandableListActivity 的用法与前面介绍的 ExpandableListView 的用法基本相似，只要为该 Activity 传入一个 ExpandableListAdapter 对象即可，接下来 ExpandableListActivity 将会生成一个显示可展开列表的窗口。

下面是 ExpandableListActivityTest 的代码。

程序清单：codes\04\4.1\OtherActivity\src\org\crazyit\app\ExpandableListActivityTest.java

```
public class ExpandableListActivityTest extends ExpandableListActivity
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ExpandableListAdapter adapter = new BaseExpandableListAdapter()
        {
            int[] logos = new int[]
            {
                R.drawable.p,
                R.drawable.z,
                R.drawable.t
            };
            private String[] armTypes = new String[]
                { "神族兵种", "虫族兵种", "人族兵种"};
            private String[][] arms = new String[][]
            {
                { "狂战士", "龙骑士", "黑暗圣堂", "电兵" },
                { "小狗", "刺蛇", "飞龙", "自爆飞机" },
                { "机枪兵", "护士MM" , "幽灵" }
            };
            //获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition, int childPosition)
            {
                return arms[groupPosition][childPosition];
            }
            @Override
            public long getChildId(int groupPosition, int childPosition)
            {
                return childPosition;
            }
            @Override
            public int getChildrenCount(int groupPosition)
            {
                return arms[groupPosition].length;
            }
            private TextView getTextView()
            {
```

```
        AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                ViewGroup.LayoutParams.FILL_PARENT, 64);
        TextView textView = new TextView(ExpandableListActivityTest.
        this);
        textView.setLayoutParams(lp);
        textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
        textView.setPadding(36, 0, 0, 0);
        textView.setTextSize(20);
        return textView;
    }
    //该方法决定每个子选项的外观
    @Override
    public View getChildView(int groupPosition, int childPosition,
            boolean isLastChild, View convertView, ViewGroup parent)
    {
        TextView textView = getTextView();
        textView.setText(getChild(groupPosition, childPosition).
        toString());
        return textView;
    }
    //获取指定组位置处的组数据
    @Override
    public Object getGroup(int groupPosition)
    {
        return armTypes[groupPosition];
    }
    @Override
    public int getGroupCount()
    {
        return armTypes.length;
    }
    @Override
    public long getGroupId(int groupPosition)
    {
        return groupPosition;
    }
    //该方法决定每个组选项的外观
    @Override
    public View getGroupView(int groupPosition, boolean isExpanded,
            View convertView, ViewGroup parent)
    {
        LinearLayout ll = new LinearLayout(ExpandableListActivity
        Test.this);
        ll.setOrientation(0);
        ImageView logo = new ImageView(ExpandableListActivityTest.
        this);
        logo.setImageResource(logos[groupPosition]);
        ll.addView(logo);
        TextView textView = getTextView();
        textView.setText(getGroup(groupPosition).toString());
        ll.addView(textView);
        return ll;
    }
    @Override
    public boolean isChildSelectable(int groupPosition, int child
    Position)
    {
        return true;
    }
    @Override
    public boolean hasStableIds()
```

```
                {
                    return true;
                }
            };
            // 设置该窗口显示列表
            setListAdapter(adapter);
        }
    }
```

上面的程序中粗体字代码为 ExpandableListActivity 设置了一个 ExpandableListAdapter 对象，即可使得该 Activity 显示可展开列表的窗口。

实例：PreferenceActivity 结合 PreferenceFragment 实现参数设置界面

PreferenceActivity 是一个非常有用的基类，当我们开发一个 Android 应用程序时，不可避免地需要进行选项设置，这些选项设置会以参数的形式保存，习惯上我们会用 Preferences 进行保存。



提示：

关于 Preferences 的介绍请参看本书第 6 章的内容。

需要指出的是，如果 Android 应用程序中包含的某个 Activity 专门用于设置选项参数，那么 Android 为这种 Activity 提供了便捷的基类：PreferenceActivity。

一旦 Activity 继承了 PreferenceActivity，那么该 Activity 完全不需要自己控制 Preferences 的读写，PreferenceActivity 会为我们处理一切。

PreferenceActivity 与普通 Activity 不同，它不再使用普通的界面布局文件，而是使用选项设置的布局文件。选项设置的布局文件以 PreferenceScreen 作为根元素——它表明定义一个参数设置的界面布局。

为了创建一个 PreferenceActivity，需要先创建一个对应的界面布局文件。从 Android 3.0 开始，Android 不再推荐直接让 PreferenceActivity 加载选项设置的布局文件。而是建议将 PreferenceActivity 与 PreferenceFragment 结合使用，其中 PreferenceActivity 只负责加载选项设置列表的布局文件，PreferenceFragment 才负责加载选项设置的布局文件。

本实例中 PreferenceActivity 加载的选项设置列表布局文件如下：

程序清单：codes\04\4.1\OtherActivity\res\xml\preference_headers.xml

```
<?xml version="1.0" encoding="utf-8"?>
<preference-headers
    xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 指定启动指定 PreferenceFragment 的列表项 -->
      <header android:fragment=
          "org.crazyit.app.PreferenceActivityTest$Prefs1Fragment"
            android:icon="@drawable/ic_settings_applications"
            android:title="程序选项设置"
            android:summary="设置应用的相关选项" />
    <!-- 指定启动指定 PreferenceFragment 的列表项 -->
      <header android:fragment=
            "org.crazyit.app.PreferenceActivityTest$Prefs2Fragment"
            android:icon="@drawable/ic_settings_display"
            android:title="界面选项设置 "
            android:summary="设置显示界面的相关选项">
            <!-- 使用 extra 可向 Activity 传入额外的数据 -->
            <extra android:name="website"
```

```
            android:value="www.crazyit.org" />
    </header>
    <!-- 使用 Intent 启动指定 Activity 的列表项 -->
    <header
        android:icon="@drawable/ic_settings_display"
        android:title="使用 Intent"
        android:summary="使用 Intent 启动某个 Activity">
        <intent android:action="android.intent.action.VIEW"
            android:data="http://www.crazyit.org" />
    </header>
</preference-headers>
```

上面的布局文件中定义了三个列表项，其中前两个列表项通过 android:fragment 选项指定启动相应的 PreferencesFragment；第三个列表项通过<intent.../>子元素启动指定的 Activity。



提示：

关于 Intent 与<intent.../>的介绍，请参考本书下一章的内容。

上面的布局文件中指定使用 Prefs1Fragment、Prefs2Fragment 两个内部类，为此我们将会在 PreferenceActivityTest 类中定义这两个内部类。下面是 PreferenceActivityTest 的代码。

程序清单：codes\04\4.1\OtherActivity\src\org\crazyit\app\PreferenceActivityTest.java

```
public class PreferenceActivityTest extends PreferenceActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 该方法用于为该界面设置一个标题按钮
        if (hasHeaders())
        {
            Button button = new Button(this);
            button.setText("设置操作");
            // 将该按钮添加到该界面上
            setListFooter(button);
        }
    }
    // 重写该该方法，负责加载页面布局文件
    @Override
    public void onBuildHeaders(List<Header> target)
    {
        // 加载选项设置列表的布局文件
        loadHeadersFromResource(R.xml.preference_headers, target);
    }
    public static class Prefs1Fragment extends PreferenceFragment
    {
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.preferences);
        }
    }
    public static class Prefs2Fragment extends PreferenceFragment
    {
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
```

```
                addPreferencesFromResource(R.xml.display_prefs);
                // 获取传入该 Fragment 的参数
                String website = getArguments().getString("website");
                Toast.makeText(getActivity()
                        , "网站域名是：" + website , Toast.LENGTH_LONG).show();
            }
        }
    }
```

上面的 Activity 重写了 PreferenceActivity 的 public void onBuildHeaders(List<Header> target)方法，重写该方法指定加载前面定义的 preference_headers.xml 界面布局文件。

上面的 Activity 中定义了两个 PreferenceFragment，它们需要分别加载 preferences.xml、display_prefs 两个选项设置的布局文件。

建立选项设置布局文件按如下步骤进行。

① 单击 Eclipse 工具条上“Opens a wizard to help create a new Android XML File”按钮（就是 ADT 插件提供的按钮中最后一个），Eclipse 弹出如图 4.2 所示的窗口。



图 4.2　创建 Android XML 文件

② 在图 4.2 所示窗口中选择创建“Preference”类型的 XML 文件，该文件默认保存在/res/xml 路径下。并选择该 XML 文件的根元素为 PreferenceScreen，单击“Finish”按钮完成创建。



提示：

上面介绍的两步就是创建Android XML文件的通用步骤，包括界面布局文件、菜单资源文件、字符串资源等，都可通过这两步来进行。

③ 单击图 4.2 所示的“Finish”按钮后进入 preferences.xml 文件（该文件默认放在 res/xml 路径下，也可以将它放入 res/layout/路径下）的编辑界面。与编辑普通界面布局文件类似，Eclipse 同样为它提供了两种编辑界面：可视化的编辑界面和源代码视图的编辑界面。通常来说，初学者可使用可视化编辑界面，但熟练用户都会比较厌烦可视化编辑界面。可视化编辑界面如图 4.3 所示。

④ 在图 4.3 所示页面的左边空白处右击鼠标，系统弹出如图 4.4 所示的快捷菜单。

⑤ 在图 4.4 所示菜单中单击“Add ...”菜单项即可开始添加界面项，如图 4.5 所示。



图 4.3　参数设置布局界面的可视化编辑



图 4.4　管理界面元素

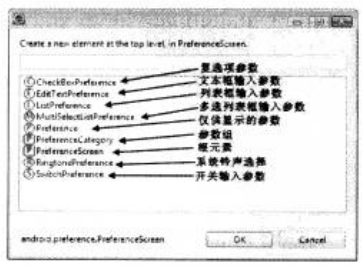


图 4.5　参数设置界面中可添加的元素

⑥ 在图 4.5 所示的界面中 PreferenceCategory 用于对参数选项进行分组，其他元素都用于设置相应的参数。单击图 4.5 所示窗口中的列表项即可添加参数设置项，重复（4）～（6）步骤，即可不断地为参数设置界面添加参数选项。填写完成后看到如下所示的界面布局文件。

程序清单：codes\04\4.1\OtherActivity\res\xml\preferences.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
<!-- 设置系统铃声 -->
<RingtonePreference
      android:ringtoneType="all"
      android:title="设置铃声"
      android:summary="选择铃声（测试 RingtonePreference）"
      android:showDefault="true"
      android:key="ring_key"
      android:showSilent="true">
</RingtonePreference>
<PreferenceCategory android:title="个人信息设置组">
      <!-- 通过输入框填写用户名 -->
      <EditTextPreference
            android:key="name"
            android:title="填写用户名"
            android:summary="填写您的用户名（测试 EditTextPreference）"
            android:dialogTitle="您所使用的用户名为：" />
      <!-- 通过列表框选择性别 -->
      <ListPreference
            android:key="gender"
            android:title="性别"
            android:summary="选择您的性别（测试 ListPreference）"
            android:dialogTitle="ListPreference"
            android:entries="@array/gender_name_list"
            android:entryValues="@array/gender_value_list" />
</PreferenceCategory>
<PreferenceCategory android:title="系统功能设置组 ">
      <CheckBoxPreference
            android:key="autoSave"
            android:title="自动保存进度"
            android:summaryOn="自动保存：开启"
            android:summaryOff="自动保存：关闭"
            android:defaultValue="true" />
</PreferenceCategory>
</PreferenceScreen>
```

上面的界面布局文件定义了一个参数设置界面，该参数设置界面中包括两个参数设置组，而且该参数设置界面全面应用了各种元素，这样方便读者以后查询。

一旦定义了参数设置的界面布局文件之后，接下来在 PreferenceFragment 程序中使用该界面布局文件进行参数设置、保存十分简单，只要如下两步即可。

① 让 Fragment 继承 PreferenceFragment。

② 在 onCreate(Bundle savedInstanceState)方法中调用 addPreferencesFromResource(..)方法加载指定的界面布局文件。

上面的实例中还用到了一个 display_prefs.xml 选项设置布局文件，该布局文件的创建步骤与 preferences.xml 文件的创建步骤相同。该文件的代码如下：

程序清单：codes\04\4.1\OtherActivity\res\xml\display_prefs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
```

```
    xmlns:android="http://schemas.android.com/apk/res/android">
<PreferenceCategory android:title="背景灯光组">
      <!-- 通过列表框选择灯光强度 -->
      <ListPreference
            android:key="light"
            android:title="灯光强度"
            android:summary="请选择灯光强度（测试ListPreference）"
            android:dialogTitle="请选择灯光强度"
            android:entries="@array/light_strength_list"
            android:entryValues="@array/light_value_list" />
</PreferenceCategory>
<PreferenceCategory android:title="文字显示组 ">
      <!-- 通过SwitchPreference设置是否自动滚屏 -->
      <SwitchPreference
            android:key="autoScroll"
            android:title="自动滚屏"
            android:summaryOn="自动滚屏：开启"
            android:summaryOff="自动滚屏：关闭"
            android:defaultValue="true" />
</PreferenceCategory>
</PreferenceScreen>
```

至此，我们为该应用程序开发了三个 Activity 类，但这三个 Activity 还不能使用，还必须在 AndroidManifest.xml 清单文件中配置 Activity 才行。

4.1.2 配置 Activity

Android 应用要求所有应用程序组件（Activity、Service、ContentProvider、BroadcastReceiver）都必须显式进行配置。

只要为<application.../>元素添加<activity.../>子元素即可配置 Activity。例如如下的配置片段：

```
<activity android:name=".SampleActivity"
      android:icon="@drawable/small_pic.png"
      android:label="@string/freneticLabel"
      android:exported="true"
      android:launchMode="singleInstance">
      ...
</activity>
```

从上面的配置片段可以看出，配置 Activity 时通常指定如下几个属性。

- name：指定该 Activity 的实现类的类名。
- icon：指定该 Activity 对应的图标。
- label：指定该 Activity 的标签。
- exported：指定该 Activity 是否允许被其他应用调用。如果将该属性设为 true，那么该 Activity 将可以被其他应用调用。
- launchMode：指定该 Activity 的加载模式，该属性支持 standard、singleTop、singleTask 和 singleInstance 这 4 种加载模式。本章后面会详细介绍这 4 种加载模式。

除此之外，配置 Activity 时通常还需要指定一个或多个<intent-filter.../>元素，该元素用于指定该 Activity 可响应的 Intent。



提示：

关于 Intent 和 IntentFilter 的介绍，请参看本书下一章的介绍。

为了在 AndroidManifest.xml 文件中配置、管理上面的三个 Activity，可以在清单文件的<application.../>元素中增加如下三个<activity.../>子元素。

程序清单：codes\04\4.1\OtherActivity\AndroidManifest.xml

```
<activity android:name=".OtherActivity"
      android:label="@string/app_name">
      <!-- 指定该 Activity 是程序的入口 -->
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
</activity>
<!-- 定义两个 Activity -->
<activity android:name=".ExpandableListActivityTest"
      android:label="查看星际兵种">
</activity>
<activity android:name=".PreferenceActivityTest"
      android:label="设置程序参数">
</activity>
```

上面的配置片段配置了三个 Activity，其中第一个 Activity 还配置了一个<intent-filter.../>元素，该元素指定该 Activity 作为应用程序的入口。

运行上面的应用程序，将看到如图 4.6 所示的界面。

在图 4.6 所示的程序界面中，用户单击任意列表项即可启动对应的 Activity，例如单击“设置程序参数”将会启动 PreferenceActivityTest。单击“查看星际兵种”将会启动 ExpandableListActivityTest。单击图 4.6 所示列表的第一个列表项将看到如图 4.7 所示的界面。



图 4.6　启动 Activity 的列表



图 4.7　选项设置列表界面

图 4.7 就是利用 PreferenceActivity 生成的选项设置列表界面。这个界面只是包含三个列表项，其中前两个列表项用于启动 PreferenceFragment，最后一个列表项将会根据 Intent 启动其他 Activity。

单击图 4.7 所示界面的第一个列表项，将可以看到如图 4.8 所示界面。

图 4.8 所示界面就是利用 PreferencesFragment 生成的选项设置界面，这个界面非常漂亮，而且系统会自动将设置的参数永久地保存到系统中——这都得益于 PreferenceActivity。例如我们单击图 4.8 所示界面的“填写用户名”列表项，系统将会显示如图 4.9 所示的输入框。



图 4.8 PreferencesFragment 生成的选项设置界面



图 4.9 使用 EditTextPreference 生成的输入框

在图 4.9 所示对话框中输入用户名，单击“OK”按钮，程序将会自动保存设置的选项参数。程序所设置的参数将会保存在/data/data/<应用程序包名>/shared_prefs 路径下，文件名为<应用程序包名>_preferences.xml。

对于本程序，运行该程序系统将会在/data/data/org.crazyit.app/shared_prefs/路径下生成一个 org.crazyit.app_preferences.xml 文件。打开 DDMS 的 File Explorer 面板，进入该参数的参数文件的保存路径，将可以看到如图 4.10 所示的界面。



图 4.10 查看系统生成的参数文件

通过图 4.10 所示窗口把其中的“org.crayit.app_preferences.xml”文件导出来，该文件的内容为：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <boolean name="autoScroll" value="true" />
    <string name="gender">medium</string>
    <boolean name="autoSave" value="true" />
    <string name="name">yeeku</string>
</map>
```

上面的文件是程序运行设置的参数。

如果单击图 4.6 所示列表框中第二个列表项，接下来将可以看到如图 4.11 所示的界面。

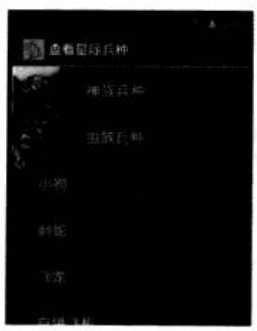


图 4.11 ExpandableListActivity 生成的可展开的列表界面

4.1.3 启动、关闭 Activity

正如前面介绍的，一个 Android 应用通常都会包含多个 Activity，但只有一个 Activity 会作为程序的入口——当该 Android 应用运行时将会自动启动并执行该 Activity。至于应用中的其他 Activity，通常都由入口 Activity 启动，或由入口 Activity 启动的 Activity 启动。

Activity 启动其他 Activity 有如下两个方法。

- startActivity(Intent intent)：启动其他 Activity。
- startActivityForResult(Intent intent, int requestCode)：以指定指定的请求码（requestCode）启动 Activity，而且程序将会等到新启动 Activity 的结果（通过重写 onActivityResult(..)方法来获取）。

启动 Activity 时可指定一个 requestCode 参数，该参数代表了启动 Activity 的请求码。这个请求码的值由开发者根据业务自行设置，用于标识请求来源。

上面两个方法都用到了 Intent 参数，Intent 是 Android 应用里各组件之间通信的重要方式，一个 Activity 通过 Intent 来表达自己“意图”——想要启动哪个组件，被启动的组件既可是 Activity 组件，也可是 Service 组件。

Android 为关闭 Activity 准备了如下两个方法。

- finish()：结束当前 Activity。
- finishActivity(int requestCode)：结束以 startActivityForResult(Intent intent, int requestCode)方法启动的 Activity。

下面的示例程序示范了如何启动 Activity，并允许程序在两个 Activity 之间切换。

程序的第一个 Activity 的界面布局很简单，该界面只包含一个按钮，该按钮用于进入第二个 Activity。此处不给出界面布局文件，Java 程序代码如下。

程序清单：codes\04\4.1\StartActivity\src\org\crazyit\app\StartActivity.java

```
public class StartActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取应用程序中的 bn 按钮
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建需要启动的 Activity 对应的 Intent
                Intent intent = new Intent(StartActivity.this, SecondActivity.
                class);
                // 启动 intent 对应的 Activity
                startActivity(intent);
            }
        });
    }
}
```

上面的程序中粗体字代码就是在 Activity 中启动其他 Activity 的关键代码。

程序中第二个 Activity 的界面同样简单：它只包含两个按钮，一个按钮用于简单地返回上一个 Activity（并不关闭自己），另一个按钮用于结束自己并返回上一个 Activity。此处不给出第二个 Activity 的界面布局文件。

下面是第二个 Activity 的 Java 代码。

程序清单：codes\04\4.1\StartActivity\src\org\crazyit\app\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        // 获取应用程序中的 previous 按钮
        Button previous = (Button) findViewById(R.id.previous);
        // 获取应用程序中的 close 按钮
        Button close = (Button) findViewById(R.id.close);
        // 为 previous 按钮绑定事件监听器
        previous.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取启动当前 Activity 的上一个 Intent
                Intent intent = new Intent(SecondActivity.this,
                        StartActivity.class);
                // 启动 intent 对应的 Activity
                startActivity(intent);
            }
        });
        // 为 close 按钮绑定事件监听器
        close.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取启动当前 Activity 的上一个 Intent
                Intent intent = new Intent(SecondActivity.this, StartActivity.
                class);
                // 启动 intent 对应的 Activity
                startActivity(intent);
                // 结束当前 Activity
                finish();
            }
        });
    }
}
```

上面的程序中两个按钮的监听器里的处理代码只有一行区别：finish()，如果有这行代码，则表明该 Activity 会结束自己。

4.1.4 使用 Bundle 在 Activity 之间交换数据

当一个 Activity 启动另一个 Activity 时，常常会有一些数据需要传过去——这就像 Web 应用从一个 Servlet 跳到另一个 Servlet 时，Web 应用习惯把需要交换的数据放入 requestScope、sessionScope 中。对于 Activity 而言，在 Activity 之间进行数据交换更简单：因为两个 Activity 之间本来就有一个“信使”：Intent，因此我们主要将需要交换的数据放入 Intent 即可。

Intent 提供了多个重载的方法来“携带”额外的数据，如下所示。

- putExtras(Bundle data)：向 Intent 中放入需要“携带”的数据包。
- Bundle getExtras()：取出 Intent 所“携带”的数据包。

- putExtra(String name, Xxx value)：向 Intent 中按 key-value 对的形式存入数据。
- getXxxExtra(String name)：从 Intent 中按 key 取出指定类型的数据。

上面方法中的 Bundle 就是一个简单的数据携带包，该 Bundle 对象包含了多个方法来存入数据。

- putXxx(String key , Xxx data)：向 Bundle 放入 Int、Long 等各种类型的数据。
- putSerializable(String key , Serializable data)：向 Bundle 中放入一个可序列化的对象。

为了取出 Bundle 数据携带包里的数据，Bundle 提供了如下方法。

- getXxx(String key)：从 Bundle 取出 Int、Long 等各种类型的数据。
- getSerializable(String key, Serializable data)：从 Bundle 取出一个可序列化的对象。

从上面的介绍不难看出，Intent 主要通过 Bundle 对象来携带数据，因此 Intent 提供了 putExtras()和 getExtras()两个方法。除此之外，Intent 也提供了多个重载的 putExtra(String name, Xxx value)、getXxxExtra(String name)，那么这些方法存、取的数据在哪里呢？其实 Intent 提供的 putExtra(String name, Xxx value)、getXxxExtra(String name)方法，只是一个便捷的方法，这些方法是直接存、取 Intent 所携带的 Bundle 中的数据。



提示：

Intent 的 putExtra(String name, Xxx value)方法是“智能”的，当程序调用 Intent 的 putExtra(String name, Xxx value)方法向 Intent 中存入数据时，如果该 Intent 中已经携带了 Bundle 对象，则该方法直接向 Intent 所携带的 Bundle 存入数据；如果 Intent 还没有携带 Bundle 对象，putExtra(String name, Xxx value)方法会先为 Intent 创建一个 Bundle，再向 Bundle 存入数据。

下面通过一个示例应用来介绍两个 Activity 之间如何通过 Bundle 交换数据。

实例：用第二个 Activity 处理注册信息

下面的程序包含两个 Activity，其中第一个 Activity 用于收集用户的输入信息，当用户单击该 Activity 的“注册”按钮时，应用进入第二个 Activity，第二个 Activity 将会获取第一个 Activity 中的数据。

下面是第一个 Activity 的界面布局文件。

程序清单：codes\04\4.1\BundleTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="请输入您的注册信息"
      android:textSize="20sp"
      />
<TableRow>
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
```

```
        android:text="用户名:"
        android:textSize="16sp"
        />
    <!-- 定义一个 EditText，用于收集用户的账号 -->
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="请填写想注册的账号"
        android:selectAllOnFocus="true"
        />
    </TableRow>
    <TableRow>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="密码:"
        android:textSize="16sp"
        />
    <!-- 用于收集用户的密码 -->
    <EditText
        android:id="@+id/passwd"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:password="true"
        android:selectAllOnFocus="true"
        />
    </TableRow>
    <TableRow>
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="性别:"
        android:textSize="16sp"
        />
    <!-- 定义一组单选框，用于收集用户注册的性别 -->
    <RadioGroup
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
    >
    <RadioButton
        android:id="@+id/male"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="男"
        android:textSize="16sp"
    />
    <RadioButton
        android:id="@+id/female"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="女"
        android:textSize="16sp"
    />
    </RadioGroup>
    </TableRow>
    <Button
        android:id="@+id/bn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="注册"
        android:textSize="16sp"
    />
    </TableLayout>
```

该界面布局对应的 Java 类代码如下。

程序清单：codes\04\4.1\BundleTest\src\org\crazyit\app\BundleTest.java

```
public class BundleTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v)
            {
                EditText name = (EditText)findViewById(R.id.name);
                EditText passwd = (EditText)findViewById(R.id.passwd);
                RadioButton male = (RadioButton) findViewById(R.id.male);
                String gender = male.isChecked() ? "男 " : "女";
                Person p = new Person(name.getText().toString(), passwd
                        .getText().toString(), gender);
                // 创建一个 Bundle 对象
                Bundle data = new Bundle();
                data.putSerializable("person", p);
                // 创建一个 Intent
                Intent intent = new Intent(BundleTest.this, ResultActivity.
                class);
                intent.putExtras(data);
                // 启动 intent 对应的 Activity
                startActivity(intent);
            }
        });
    }
}
```



图 4.12　注册界面

上面的程序中粗体字代码根据用户输入创建了一个 Person 对象，Person 类只是一个简单的 DTO 对象，该 Person 类实现了 java.io.Serializable 接口，因此 Person 对象是可序列化的。

上面的程序创建了一个 Bundle 对象，并调用 putSerializable("person", p)将 Person 对象放入该 Bundle 中，然后再使用 Intent 来“携带”这个 Bundle，这样即可将 Person 对象传入第二个 Activity。

运行该程序，第一个 Activity 显示的界面如图 4.12 所示。

当用户单击图 4.12 的“注册”按钮时，程序将会启动 ResultActivity，并将用户输入的数据传入该 Activity。下面是 ResultActivity 的界面布局文件。

程序清单：codes\04\4.1\BundleTest\res\layout\result.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    >
```

```
<!-- 定义三个 TextView，用于显示用户输入的数据 -->
<TextView
      android:id="@+id/name"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textSize="18sp"
      />
<TextView
      android:id="@+id/passwd"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textSize="18sp"
      />
<TextView
      android:id="@+id/gender"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:textSize="18sp"
      />
</LinearLayout>
```

这个 Activity 的程序将会从 Bundle 中取出前一个 Activity 传过来的数据，并将它们显示出来。该 Activity 的 Java 代码如下。

程序清单：codes\04\4.1\BundleTest\src\org\crazyit\app\ResultActivity.java

```
public class ResultActivity extends Activity
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.result);
            TextView name = (TextView) findViewById(R.id.name);
            TextView passwd = (TextView) findViewById(R.id.passwd);
            TextView gender = (TextView) findViewById(R.id.gender);
            // 获取启动该 Result 的 Intent
            Intent intent = getIntent();
            // 直接通过 Intent 取出它所携带的 Bundle 数据包中的数据
            Person p = (Person) intent.getSerializableExtra("person");
            name.setText("您的用户名为：" + p.getName());
            passwd.setText("您的密码为：" + p.getPass());
            gender.setText("您的性别为：" + p.getGender());
      }
}
```

上面的程序中粗体字代码用于获取前一个 Activity 所传过来的数据，至于该 Activity 获取数据之后如何处理它们，完全由开发者自己决定。本应用程序只是将获取的数据显示出来，用户在图 4.12 所示界面中输入注册信息之后，单击“注册”按钮将看到如图 4.13 所示的界面。



图 4.13 注册成功

➤➤4.1.5 启动其他 Activity 并返回结果

前面已经提到，Activity 还提供了一个 startActivityForResult(Intent intent, int requestCode) 方法来启动其他 Activity。该方法用于启动指定 Activity，而且期望获取指定 Activity 返回的结果。这种请求对于实际应用也是很常见的，例如应用程序第一个界面需要用户进行选择——但需要选择的列表数据比较复杂，必须启动另一个 Activity 让用户选择。当用户在第二个

Activity 选择完成后，程序返回第一个 Activity，第一个 Activity 必须能获取并显示用户在第二个 Activity 选择的结果。在这种应用场景下，也是通过 Bundle 进行数据交换的。

为了获取被启动的 Activity 所返回的结果，需要从两方面着手：

- 当前 Activity 需要重写 onActivityResult(int requestCode , int resultCode , Intent intent)，当被启动的 Activity 返回结果时，该方法将会被触发，其中 requestCode 代表请求码，而 resultCode 代表 Activity 返回的结果码，这个结果码也是由开发者根据业务自行设定的。
- 被启动的 Activity 需要调用 setResult()方法设置处理结果。

一个 Activity 中可能包含多个按钮，并调用多个 startActivityForResult()方法来打开多个不同的 Activity 处理不同的业务，当这些新 Activity 关闭后，系统都将回调前面 Activity 的 onActivityResult(int requestCode, int resultCode, Intent data)方法。为了知道该方法是由哪个请求的结果所触发的，可利用 requestCode 请求码；为了知道返回的数据来自于哪个新的 Activity，可利用 resultCode 结果码。

下面通过一个示例来介绍如何启动 Activity 并获取被启动 Activity 的结果。

实例：用第二个 Activity 让用户选择信息

下面的程序也包含两个 Activity，第一个 Activity 的界面布局比较简单，它只包含一个按钮和一个文本框，故此处不再给出界面布局文件。第一个 Activity 对应的 Java 代码如下。

程序清单：codes\04\4.1\ActivityForResult\src\org\crazyit\app\ActivityForResult.java

```
public class ActivityForResult extends Activity
{
    Button bn;
    EditText city;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面上的组件
        bn = (Button) findViewById(R.id.bn);
        city = (EditText) findViewById(R.id.city);
        // 为按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建需要对应于目标 Activity 的 Intent
                Intent intent = new Intent(ActivityForResult.this,
                        SelectCityActivity.class);
                // 启动指定 Activity 并等待返回的结果，其中 0 是请求码，用于标识该请求
                startActivityForResult(intent, 0);
            }
        });
    }
    ...
}
```

上面的程序中粗体字代码用于启动 ActivityForResult，并等待该 Activity 返回的结果——但问题是：该 Activity 如何获取 ActivityForResult 返回的结果呢？当前 Activity 启动

SelectCityActivity 之后，SelectCityActivity 何时返回结果是不确定的，因此当前 Activity 无法去获取 SelectCityActivity 返回的结果。

为了让当前 Activity 获取 SelectCityActivity 所返回的结果，开发者应该重写 onActivityResult()方法——当被启动的 SelectCityActivity 返回结果时，onActivityResult()方法将会被回调。

因此还需要为上面的 ActivityForResult 添加如下方法。

程序清单：codes\04\4.1\ActivityForResult\src\org\crazyit\activity\ActivityForResult.java

```
// 重写该方法，该方法以回调的方式来获取指定 Activity 返回的结果
@Override
public void onActivityResult(int requestCode
    , int resultCode, Intent intent)
{
    // 当 requestCode、resultCode 同时为 0 时，也就是处理特定的结果
    if (requestCode == 0 && resultCode == 0)
    {
        // 取出 Intent 里的 Extras 数据
        Bundle data = intent.getExtras();
        // 取出 Bundle 中的数据
        String resultCity = data.getString("city");
        // 修改 city 文本框的内容
        city.setText(resultCity);
    }
}
```

运行该程序，将看到如图 4.14 所示的界面。

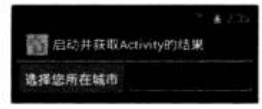


图 4.14 让用户选择所在城市

单击图 4.14 所示界面中“选择您所在城市”按钮，系统将会启动 SelectCityActivity，该 SelectCityActivity 将会显示一个可展开的列表。该 SelectCityActivity 无须界面布局文件。该 SelectCityActivity 的 Java 代码如下。

程序清单：codes\04\4.1\ActivityForResult\src\org\crazyit\app\SelectCityActivity.java

```
public class SelectCityActivity extends ExpandableListActivity
{
    // 定义省份数组
    private String[] provinces = new String[]
    { "广东", "广西", "湖南"};
    private String[][] cities = new String[][]
    {
        { "广州", "深圳", "珠海", "中山" },
        { "桂林", "柳州", "南宁", "北海" },
        { "长沙", "岳阳" , "衡阳" , "株洲" }
    };
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ExpandableListAdapter adapter = new BaseExpandableListAdapter()
        {
            // 获取指定组位置、指定子列表项处的子列表项数据
            @Override
            public Object getChild(int groupPosition, int childPosition)
            {
                return cities[groupPosition][childPosition];
            }
            @Override
            public long getChildId(int groupPosition, int childPosition)
```

```
{
    return childPosition;
}
@Override
public int getChildrenCount(int groupPosition)
{
    return cities[groupPosition].length;
}
private TextView getTextView()
{
    AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
            ViewGroup.LayoutParams.MATCH_PARENT, 64);
    TextView textView = new TextView(SelectCityActivity.this);
    textView.setLayoutParams(lp);
    textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
    textView.setPadding(36, 0, 0, 0);
    textView.setTextSize(20);
    return textView;
}
// 该方法决定每个子选项的外观
@Override
public View getChildView(int groupPosition, int childPosition,
        boolean isLastChild, View convertView, ViewGroup parent)
{
    TextView textView = getTextView();
    textView.setText(getChild(groupPosition, childPosition)
            .toString());
    return textView;
}
// 获取指定组位置处的组数据
@Override
public Object getGroup(int groupPosition)
{
    return provinces[groupPosition];
}
@Override
public int getGroupCount()
{
    return provinces.length;
}
@Override
public long getGroupId(int groupPosition)
{
    return groupPosition;
}
// 该方法决定每个组选项的外观
@Override
public View getGroupView(int groupPosition, boolean isExpanded,
        View convertView, ViewGroup parent)
{
    LinearLayout ll = new LinearLayout(SelectCityActivity.this);
    ll.setOrientation(0);
    ImageView logo = new ImageView(SelectCityActivity.this);
    ll.addView(logo);
    TextView textView = getTextView();
    textView.setText(getGroup(groupPosition).toString());
    ll.addView(textView);
    return ll;
}
@Override
public boolean isChildSelectable(int groupPosition,
```

```
                    int childPosition)
            {
                return true;
            }
            @Override
            public boolean hasStableIds()
            {
                return true;
            }
        };
        // 设置该窗口显示列表
        setListAdapter(adapter);
        getExpandableListView().setOnChildClickListener(
            new OnChildClickListener()
            {
                @Override
                public boolean onChildClick(ExpandableListView parent,
                        View source, int groupPosition, int childPosition,
                        long id)
                {
                    // 获取启动该Activity之前的Activity对应的Intent
                    Intent intent = getIntent();
                    intent.putExtra("city",
                            cities[groupPosition][childPosition]);
                    // 设置该SelectActivity的结果码,并设置结束之后退回的Activity
                    SelectCityActivity.this.setResult(0, intent);
                    // 结束SelectCityActivity。
                    SelectCityActivity.this.finish();
                    return false;
                }
            });
    }
}
```

上面的 Activity 只是一个普通的显示可展开列表的 Activity，程序还为该 Activity 的各子列表项绑定了事件监听器，当用户单击子列表项时，该 Activity 将会把用户选择城市返回给上一个 Activity。

当上一个 Activity 获取 SelectCityActivity 选择城市之后，将会把该程序显示在图 4.14 所示界面右边的文本框内。

4.2 Activity 的回调机制

有 Web 开发经验的读者都知道：当一个 Servlet 开发出来之后，该 Servlet 运行于 Web 服务器中。服务器何时创建 Servlet 的实例，何时回调 Servlet 的方法向用户生成响应，程序员无法控制，这种回调由服务器自行决定。

前面已经提到：Android 应用中的 Activity 与 Web 应用中的 Servlet 有点相似，也就是说，Activity 被开发出来之后，开发者只要在 AndroidManifest.xml 文件配置该 Activity 即可。至于该 Activity 何时被实例化，它的方法何时被调用，对开发者来说是完全透明的。

当开发者开发一个 Servlet 时，根据不同需求场景，可能需要选择性地实现如下方法：

- init(ServletConfig config)
- destroy()
- doGet(HttpServletRequest req, HttpServletResponse resp)

- doPost(HttpServletRequest req, HttpServletResponse resp)
- service(HttpServletRequest req, HttpServletResponse resp)

当把这个 Servlet 部署在 Web 应用中之后，Web 服务器将会在特定的时刻，调用该 Servlet 上面的各种方法——这种调用就被称为所谓的回调。

所谓回调，在实现具有通用性质的应用架构时非常常见：对于一个具有通用性质的程序架构来说，程序架构完成整个应用的通用功能、流程，但在某个特定的点上，需要一段业务相关的代码——通用的程序架构无法实现这段代码，那么程序架构会在这个点上留一个“空”。

对于 Java 程序来说，程序架构在某个点上留的“空”，可以以如下两种方式存在。

- 以接口形式存在：该接口由开发者实现，实现该接口时将会实现该接口的方法，那么通用的程序架构就会回调该方法来完成业务相关的处理。
- 以抽象方法（也可以是非抽象方法）的形式存在：这就是 Activity 的实现形式。这些特定的点上方法已经被定义了，如 onCreate、onActivityResult 等方法，开发者可以选择性地重写这些方法，通用的程序架构就会回调该方法来完成业务相关的处理。



提示：

回调机制的第一个种实现方式就是典型的命令者模式，关于命令者模式请参考疯狂 Java 体系的《轻量级 Java EE 企业应用实战》第 9 章。

前面介绍的事件处理也用到了回调机制：当开发者开发一个组件时，如果开发者需要该组件能响应特定的事件，可以选择性地实现该组件的特定方法——当用户在该组件上激发某个事件时，该组件上特定的方法就会回调。

Activity 的回调机制也与此类似，当 Activity 被部署在 Android 应用中之后，随着应用程序的运行，Activity 会不断地在不同的状态之间切换，该 Activity 中特定的方法就会被回调——开发者就可以选择性地重写这些方法来加入业务相关的处理。

Activity 运行过程所处的不同状态也被称为生命周期，下面将详细介绍 Activity 的生命周期。

4.3 Activity 的生命周期与加载模式

当 Activity 处于 Android 应用中运行时，它的活动状态由 Android 以 Activity 栈的形式管理。当前活动的 Activity 位于栈顶。随着不同应用的运行，每个 Activity 都有可能从活动状态转入非活动状态，也可能从非活动状态转入活动状态。

4.3.1 Activity 的生命周期演示

归纳起来 Activity 大致会经过如下 4 个状态。

- 活动状态：当前 Activity 位于前台，用户可见，可以获得焦点。
- 暂停状态：其他 Activity 位于前台，该 Activity 依然可见，只是不能获得焦点。
- 停止状态：该 Activity 不可见，失去焦点。
- 销毁状态：该 Activity 结束，或 Activity 所在的 Dalvik 进程被结束。

图 4.15（该图参照 Android 官方文档，但细节方面比官方文档更准确）显示了 Activity

生命周期及相关回调方法。

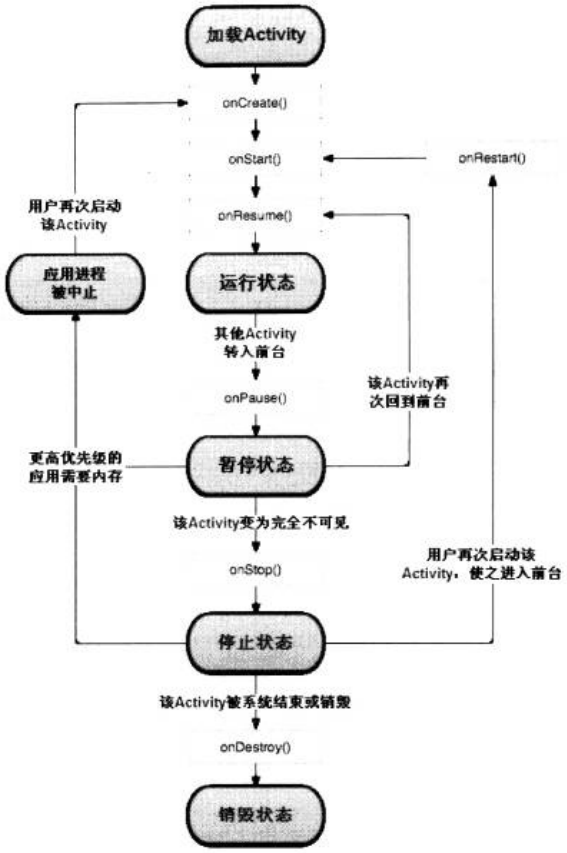


图 4.15 Activity 生命周期及回调方法

从图 4.15 可以看出，在 Activity 的生命周期中，如下方法会被系统回调。

- onCreate(Bundle savedStatus)：创建 Activity 时被回调。该方法只会被调用一次。
- onStart()：启动 Activity 时被回调。
- onRestart()：重新启动 Activity 时被回调。
- onResume()：恢复 Activity 时被回调，onStart()方法后一定会回调 onResume()方法。
- onPause()：暂停 Activity 时被回调。
- onStop()：停止 Activity 时被回调。
- onDestroy()：销毁 Activity 时被回调。该方法只会被调用一次。

正如开发 Servlet 时可以根据需要选择性地覆盖指定方法，开发 Activity 时也可根据需要选择性地覆盖指定方法。其中最常见的就是覆盖 onCreate(Bundle saveStatus)方法——前面所有示例都覆盖了 Activity 的 onCreate(Bundle saveStatus)方法，该方法用于对该 Activity 执行初始化。除此之外，覆盖 onPause()方法也很常见：比如用户正在玩一个游戏，此时有电话进来，那么我们需要将当前（游戏）暂停，并保存该游戏的进行状态，这就可以通过覆盖 onPause()方法来实现。接下来当用户再次切换到游戏状态时，onResume()方法已经会被回调，因此可以通过重写 onResume()方法来恢复游戏状态。

下面的 Activity 覆盖了上面的 7 个生命周期方法，并在每个方法中增加了一行记录日志

代码。该 Activity 的界面布局很简单，包含了 2 个按钮：一个用于启动一个对话框风格的 Activity，另一个用于退出该应用，此处不给出界面布局代码。该 Activity 的 Java 代码如下。

程序清单：codes\04\4.3\Lifecycle\src\org\crazyit\app\Lifecycle.java

```
public class Lifecycle extends Activity
{
    final String TAG = "--CrazyIt--";
    Button finish ,startActivity;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 输出日志
        Log.d(TAG, "-------onCreate------");
        finish = (Button) findViewById(R.id.finish);
        startActivity = (Button) findViewById(R.id.startActivity);
        // 为 startActivity 按钮绑定事件监听器
        startActivity.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                Intent intent = new Intent(Lifecycle.this
                        , SecondActivity.class);
                startActivity(intent);
            }
        });
        // 为 finish 按钮绑定事件监听器
        finish.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 结束该 Activity
                Lifecycle.this.finish();
            }
        });
    }
    @Override
    public void onStart()
    {
        super.onStart();
        // 输出日志
        Log.d(TAG, "-------onStart------");
    }
    @Override
    public void onRestart()
    {
        super.onRestart();
        // 输出日志
        Log.d(TAG, "-------onRestart------");
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 输出日志
        Log.d(TAG, "-------onResume------");
    }
    @Override
```

```
    public void onPause()
    {
        super.onPause();
        // 输出日志
        Log.d(TAG, "-------onPause------");
    }
    @Override
    public void onStop()
    {
        super.onStop();
        // 输出日志
        Log.d(TAG, "-------onStop------");
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 输出日志
        Log.d(TAG, "-------onDestroy------");
    }
}
```

将该 Activity 设置成程序的入口 Activity，当程序启动时将会自动启动并执行该 Activity，此时将可以在 DDMS 的 LogCat 窗口看到如图 4.16 所示的输出。



图 4.16 启动 Activity 时回调的方法

单击该程序界面上“启动对话框风格的 Activity”按钮，此时对话框风格的 Activity 进入前台，但 Lifecycle 虽然不能获得焦点，但依然“部分可见”，此时该 Activity 进入“暂停”状态。此时将可以在 DDMS 的 LogCat 窗口看到如图 4.17 所示的输出。



图 4.17 暂停 Activity 时回调的方法

在当前状态下，按下模拟器右边的键，再次返回 Lifecycle Activity，该 Activity 再次进入“运行”状态，此时看到 LogCat 有如图 4.18 所示的输出。



图 4.18 恢复 Activity 时回调的方法

在当前程序运行状态下，按下模拟器右边的键，返回系统桌面，当前该 Activity 将失去焦点且不可见，但该 Activity 并未被销毁，它进入“停止”状态。此时看到 LogCat 有如图 4.19 所示的输出。



图 4.19　停止 Activity 时回调的方法

在模拟器程序列表处再次找到该应用程序并启动它，将可以看到 DDMS 的 LogCat 窗口有如图 4.20 所示的输出。



图 4.20　重新启动 Activity 时回调的方法

如果用户单击该程序界面上的“退出”按钮，该 Activity 将会结束自己，并且可以在 DDMS 的 LogCat 窗口看到如图 4.21 所示的输出。



图 4.21　结束 Activity 时回调的方法

通过上面的运行过程，相信读者对 Activity 的生命周期状态及在不同状态之间切换时所回调的方法有了很清晰的认识。

4.3.2　Activity 与 Servlet 的相似性与区别

虽然很少有人会把 Activity 和 Servlet 放在一起对比，但就笔者的经验来看，Activity 与 Servlet 之间确实存在不少相似之处，如果读者已经具备一定的 Web 开发经验，通过这种“触类旁通”式的学习，相信可以更好地理解 Activity 的设计思想。



提示：笔者一直坚信“温故而知新”是一种好的学习方式：当我们学习一门新的知识时，最好能找到新知识和已掌握知识之间的类比关系，这样既可迅速获得对新知识的直观把握，又可巩固已掌握的旧知识，避免“知识越学越多”的困扰。你

的知识越积累越多，你会发现眼界越来越高，视野越来越广，看问题更容易简明、扼要地把握本质。

当然，两个知识之间除了存在相似的地方之外，也少不了差异，这部分差异正是需互相参考、互相对照的部分，以求深入理解各自的设计思想、原理。当我们学习知识时，除了掌握怎么用它之外，最好能站在设计者的角度来看：他为何要设计这个类？这个类为何要包含这些方法？方法为何要有这些形参？理解这些设计，剩下的也就简单了。

Activity 与 Servlet 的相似之处大致如下：

- Activity、Servlet 的职责都是向用户呈现界面。
- 开发者开发 Activity、Servlet 都继承系统的基类。
- Activity、Servlet 开发出来之后都需要进行配置。
- Activity 运行于 Android 应用中，Servlet 运行于 Web 应用中。
- 开发者无须创建 Activity、Servlet 的实例，无须调用它们的方法。Activity、Servlet 的方法都由系统以回调的方式来调用。
- Activity、Servlet 都有各自的生命周期，它们的生命周期都由外部负责管理。
- Activity、Servlet 都不会直接相互调用，因此都不能直接进行数据交换。Servlet 之间的数据交换需要借助于 Web 应用提供的 requestScope、sessionScope 等；Activity 之间的数据交换要借助于 Bundle。

当然，Activity 与 Servlet 之间的差别很多，因为它们本身所在场景是完全不同的，它们之间的区别也很明显：

- Activity 是 Android 窗口的容器，因此 Activity 最终以窗口的形式显示出来。而 Servlet 并不会生成应用界面，只是向浏览者生成文本响应。
- Activity 运行于 Android 应用中，因此 Activity 的本质还是通过各种界面组件来搭建界面；而 Servlet 则主要以 IO 流向浏览者生成文本响应，浏览者看到的界面其实是由浏览器负责生成的。
- Activity 之间的跳转主要通过 Intent 对象来控制；而 Servlet 之间的跳转则主要由用户请求来控制。

4.3.3 Activity 的 4 种加载模式

正如前面介绍 Activity 配置时提到的，配置 Activity 时可指定 android:launchMode 属性，该属性用于配置该 Activity 的加载模式，该属性支持如下 4 个属性值。

- standard：标准模式，这是默认的加载模式。
- singleTop：Task 顶单例模式。
- singleTask：Task 内单例模式。
- singleInstance：全局单例模式。

可能有读者会问：为什么要为 Activity 指定加载模式？加载模式有什么用？在讲解 Activity 的加载模式之前，先介绍 Android 对 Activity 的管理：Android 采用 Task 来管理多个 Activity，当我们启动一个应用时，Android 就会为之创建了一个 Task，然后启动这个应用的

入口 Activity（即<intent-filter.../>中配置为 MAIN 和 LAUNCHER 的 Activity）。

Android 的 Task 是一个有点麻烦的概念——因为 Android 并没有为 Task 提供 API，因此开发者无法真正去访问 Task，只能调用 Activity 的 getTaskId()方法来获取它所在的 Task 的 ID。事实上，我们可以把 Task 理解成 Activity 栈，Task 以栈的形式来管理 Activity：先启动的 Activity 被放在 Task 栈底，后启动的 Activity 被放在 Task 栈顶。

那么 Activity 的加载模式，就负责管理实例化、加载 Activiyt 的方式、并可以控制 Activity 与 Task 之间的加载关系。

下面详细介绍这 4 种加载模式。

1. standard 模式

每次通过这种模式来启动目标 Activity 时，Android 总会为目标 Activity 创建一个新的实例，并将该 Activity 添加到当前 Task 栈中——这种模式不会启动新的 Task，新 Activity 将被添加到原有的 Task 中。

下面的示例使用了 standard 模式来不断启动自身。

程序清单：codes\04\4.3\StandardTest\src\org\crazyit\activity\StandardTest.java

```
public class StandardTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为: " + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 StandardTest");
        // 添加 TextView 和 Button
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，当单击该按钮时启动 StandardTest
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建启动 StandardTest 的 Intent
                Intent intent = new Intent(StandardTest.this
                    , StandardTest.class);
                startActivity(intent);
            }
        });
    }
}
```

正如上面的粗体字代码所示，每次单击按钮，程序将会再次启动 StandardTest Activity，程序配置该 Activity 时无须指定 launchMode 属性，该 Activity 默认采用 standard 加载模式。

运行该程序，多次单击程序界面上的“启动 StandardTest”按钮，程序将会不断启动新的 StandardTest 实例（不同 Activity 实例的 hashCode 值有差异），但它们所在的 Task ID 总是相

同的——这表明这种加载模式不会使用全新的 Task。

standard 加载模式的示意图如图 4.22 所示。

正如图 4.22 所示，当用户单击手机的“返回”键时，系统将会“逐一”从 Activity 栈顶删除 Activity 实例。

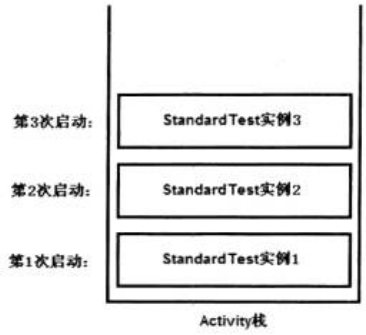


图 4.22 standard 加载模式

2. singleTop 模式

这种模式与 standard 模式基本相似，但有一点不同：当将要被启动的目标 Activity 已经位于 Task 栈顶时，系统不会重新创建目标 Activity 的实例，而是直接复用已有的 Activity 实例。

如果将上面实例中 StandardTest Activity 的加载模式改为 standard 加载模式，无论用户点击多少次按钮，界面上的程序将不会有任何变化。

如果将要被启动的目标 Activity 没有位于 Task 栈顶，此时系统会重新创建目标 Activity 的实例，并将它加载到 Task 的栈顶——此时与 standard 模式完全相同。

3. singleTask 模式

采用这种加载模式的 Activity 在同一个 Task 内只有一个实例，当系统采用 singleTask 模式启动目标 Activity 时，可分为如下三种情况：

- 如果将要启动的目标 Activity 不存在，系统将会创建目标 Activity 的实例，并将它加入 Task 栈顶。
- 如果将要启动的目标 Activity 已经位于 Task 栈顶，此时与 singleTop 模式的行为相同。
- 如果将要启动的目标 Activity 已经存在、但没有位于 Task 栈顶，系统将会把位于该 Activity 上面的所有 Activity 移出 Task 栈，从而使得目标 Activity 转入栈顶。

下面的示例示范了上面第三种情形，该实例包含两个 Activity，其中第一个 Activity 上显示文本框和按钮，该按钮用于启动启动第二个 Activity；第二个 Activity 上显示文本框和按钮，该按钮用于启动第一个 Activity。

第一个 Activity 的代码如下。

程序清单：codes\04\4.3\SingleTaskTest\src\org\crazyit\activity\SingleTaskTest.java

```
public class SingleTaskTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为: " + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SecondActivity");
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，当单击该按钮时启动 SecondActivity
```

```
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(SingleTaskTest.this
                    , SecondActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

正如上面的粗体字代码所示，当用户单击该 Activity 上的按钮时，系统将会启动 SecondActivity，下面是 SecondActivity 的代码。

程序清单：codes\04\4.3\SingleTaskTest\src\org\crazyit\activity\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为：" + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SingleTaskTest");
        // 添加 TextView 和 Button
        layout.addView(button);
        layout.addView(tv);
        // 为 button 添加事件监听器，当单击该按钮时启动 SingleTaskTest
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(SecondActivity.this,
                    SingleTaskTest.class);
                startActivity(intent);
            }
        });
    }
}
```

正如上面的粗体字代码所示，当用户单击该 Activity 上的按钮时，系统将会启动 SingleTaskTest。程序将该 SecondActivity 的加载模式配置成 singleTask。运行该示例，系统默认启动 SingleTaskTest Activity，单击该界面上的按钮，系统将以 singleTask 模式打开 SecondActivity，如图 4.23 所示：



图 4.23　singleTask 模式

图 4.23 所示界面的 Task 栈中目前有两个 Activity（从底向上）：SingleTaskTest→SecondActivity。

单击图 4.23 所示界面上的“启动 SingleTaskTest”按钮，系统以

标准模式再次加载一个新的 SingleTaskTest Activity。

此时 Task 栈中有三个 Activity（从底向上）：SingleTaskTest → SecondActivity → SingleTaskTest。在 SingleTaskTest 的界面上再次单击按钮，系统将会以 singleTask 模式再次打开 SecondActivity，系统会将位于 SecondActivity 上面的所有 Activity 移出，使得 SecondActivity 进入栈顶。此时 Task 栈中只有两个 Activity（从底向上）：SingleTaskTest→SecondActivity，也就是再次恢复到图 4.23 所示的状态。

4. singleInstance 模式

这种加载模式下，系统保证无论从哪个 Task 中启动目标 Activity，只会创建一个目标 Activity 实例，并会使用一个全新的 Task 栈来装载该 Activity 实例。

当系统采用 singleInstance 模式启动目标 Activity 时，可分为如下两种情况：

- 如果将要启动的目标 Activity 不存在，系统会先创建一个全新的 Task、再创建目标 Activity 的实例，并将它加入新的 Task 的栈顶。
- 如果将要启动的目标 Activity 已经存在，无论它位于哪个应用程序中，无论它位于哪个 Task 中，系统将会把该 Activity 所在的 Task 转到前台，从而使用该 Activity 显示出来。

需要指出的是，采用 singleInstance 模式加载 Activity 总是位于 Task 栈顶，采用 singleInstance 模式加载 Activity 所在 Task 只包含该 Activity。

下面示例中的 SingleInstanceTest 中包含一个按钮，当用户单击该按钮时，系统启动 SecondActivity，程序清单如下。

程序清单：codes\04\4.3\SingleInstanceTest\src\org\crazyit\activity\SingleInstanceTest.java

```
public class SingleInstanceTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为: " + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SecondActivity");
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，当单击该按钮时启动 SecondActivity
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                Intent intent = new Intent(SingleInstanceTest.this
                    , SecondActivity.class);
                startActivity(intent);
            }
        });
    }
}
```

上面的粗体字代码指定单击按钮时将会启动 SecondActivity，将该 SecondActivity 配置成 singleInstance 加载模式，并且将该 Activity 的 exported 属性配置成 true——表明该 Activity 可被其他应用启动。

配置该 Activity 的配置片段如下：

```
<activity android:name=".SecondActivity"
    android:label="@string/second"
    android:exported="true"
    android:launchMode="singleInstance">
    <intent-filter>
        <!-- 指定该 Activity 能响应 Action 为指定字符串的 Intent -->
        <action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

配置该 Activity 时，将它的 exported 属性设 true，表明允许通过其他程序来启动该 Activity；配置该 Activity 时还配置了<intent-filter.../>子元素，这表明该 Activity 可通过隐式 Intent 启动——关于隐式 Intent 介绍，请参考下一章的内容。

运行该示例，系统默认显示 SingleInstanceTest，当用户单击该 Activity 界面上的按钮时，系统将会采用 singleInstance 模式加载 SecondActivity：系统启动新的 Task、并用新的 Task 加载新创建的 SecondActivity 实例，SecondActivity 总是位于该新 Task 的栈顶。此时可看到如图 4.24 所示界面。



图 4.24　singleIntance 加载模式

另一个示例将采用隐式 Intent 再次启动该 SecondActivity，下面是采用隐式 Intent 启动 SecondActivity 的示例的代码。

程序清单：codes\04\4.3\OtherTest\src\org\crazyit\other\OtherTest.java

```
public class OtherTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);
        this.setContentView(layout);
        // 创建一个 TextView 来显示该 Activity 和它所在 Task ID
        TextView tv = new TextView(this);
        tv.setText("Activity 为：" + this.toString()
            + "\n" + ", Task ID 为:" + this.getTaskId());
        Button button = new Button(this);
        button.setText("启动 SecondActivity");
        // 添加 TextView 和 Button
        layout.addView(tv);
        layout.addView(button);
        // 为 button 添加事件监听器，使用隐式 Intent 启动目标 Activity
        button.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 使用隐式 Intent 启动 Second Activity
                Intent intent = new Intent();
```

```
        intent.setAction("org.crazyit.intent.action.CRAZYIT_ACTION");
                        startActivity(intent);
                    }
                });
        }
}
```

运行该示例，系统默认显示 OtherTest，当用户单击该 Activity 界面上的按钮时，系统将采用隐式 Intent 来启动 SecondActivity，注意 SecondActivity 的加载模式是 singleInstance，如果前一个示例还未退出，无论 SecondActivity 所在 Task 是否位于前台，系统将再次把 SecondActivity 所在 Task 转入前台，从而将 SecondActivity 显示出来。也就是说，系统将会再次显示图 4.24 所示的界面。

4.4 Fragment 详解

第 2 章介绍 ActionBar 时已经用到了 Fragment，Fragment 是 Android 3.0 引入的新 API。Fragment 代表了 Activity 的子模块，因此可以把 Fragment 理解成 Activity 片段（Fragment 本身就是片段的意思）。Fragment 用于自己的生命周期，也可以接受它自己的输入事件。

4.4.1 Fragment 概述及其设计哲学

Fragment 必须被“嵌入”Activity 中使用，因此虽然 Fragment 也拥有自己的生命周期，但 Fragment 的生命周期会受它所在的 Activity 的生命周期的控制。例如，当 Activity 暂停时，该 Activity 内的所有 Fragment 都会暂停；当 Activity 被销毁时，该 Activity 内的所有 Fragment 都会被销毁。只有当该 Activity 处于活动状态时，程序员可通过方法独立地操作 Fragment。

关于 Fragment，可以归纳出如下几个特征：

- Fragment 总是作为 Activity 界面的组成部分。Fragment 可调用 getActivity()方法获取它所在的 Activity，Activity 可调用 FragmentManager 的 findFragmentById()或 findFragmentByTag()方法来获取 Fragment。
- 在 Activity 运行过程中，可调用 FragmentManager 的 add()、remove()、replace()方法动态地添加、删除或替换 Fragment。
- 一个 Activity 可以同时组合多个 Fragment；反过来，一个 Fragment 也可被多个 Activity 复用。
- Fragment 可以响应自己的输入事件，并拥有自己的生命周期，但它们的生命周期直接被其所属的 Activity 的生命周期控制。

Android 3.0 引入 Fragment 的初衷是为了适应大屏幕的平板电脑，由于平板电脑的屏幕比手机屏幕更大，因此可以容纳更多的 UI 组件，且这些 UI 组件之间存在交互关系。Fragment 简化了大屏幕 UI 的设计，它不需要开发者管理组件包含关系的复杂变化，开发者使用 Fragment 对 UI 组件进行分组、模块化管理，可以更方便地在运行过程中动态更新 Activity 的用户界面。

例如有如下新闻浏览界面，该界面需要在屏幕左边显示新闻列表，并在屏幕右边显示新闻内容，此时就可以在 Activity 中显示两个并排的 Fragment：左边的 Fragment 显示新闻列表，右边的 Fragment 显示新闻内容。由于每个 Fragment 拥有自己的生命周期，并可响应用户输

入事件，因此可以非常方便地实现：当用户单击左边列表的指定新闻时，右边的 Fragment 显示相应的新闻内容。图 4.25 左边的“平板电脑”部分显示了这种 UI 界面。

通过使用上面的 Fragment 设计机制，可以取代传统的让一个 Activity 显示新闻列表，另一个 Activity 显示新闻内容的设计。

由于 Fragment 是可复用的组件，因此如果需要在正常尺寸的手机屏幕上运行该应用，可以改为使用两个 Activity：ActivityA 包含 FragmentA、ActivityB 包含 FragmentB。其中 ActivityA 仅包含显示文章列表 FragmentA，而当用户选择一篇文章时，它会启动包含新闻内容的 ActivityB，如图 4.25 右边的“手机”部分。由此可见，Fragment 可以很好地支持图 4.25 所示的两种设计模式。

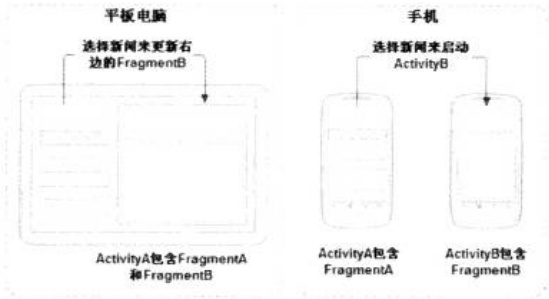


图 4.25　Fragment 的设计哲学

4.4.2　创建 Fragment

与创建 Activity 类似，开发者实现的 Fragment 必须继承 Fragment 基类，Android 提供了如图 4.26 所示的 Fragment 继承体系。

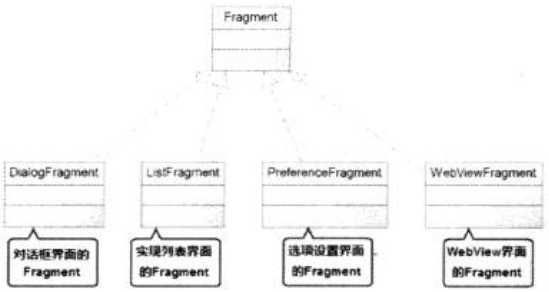


图 4.26　Fragment 继承体系

开发者实现的 Fragment，可以根据需要继承图 4.26 所示的 Fragment 基类或它的任意子类。接下来，实现 Fragment 与实现 Activity 非常相似，它们都需要实现与 Activity 类似的回调方法，例如 onCreate()、onCreateView()、onStart()、onResume()、onPause()、onStop()等。



提示：开发 Fragment 与开发 Activity 非常相似，区别只是开发 Activity 需要继承 Activity 或其子类；但开发 Fragment 需要继承 Fragment 及其子类。与此同时，只要将原来写在 Activity 回调方法的代码“移到”Fragment 的回调方法中即可。

通常来说，创建 Fragment 通常需要实现如下三个方法。

- onCreate()：系统创建 Fragment 对象后回调该方法，实现代码中只初始化想要在 Fragment 中保持的必要组件，当 fragment 被暂停或者停止后可以恢复。
- onCreateView()：当 Fragment 绘制界面组件时会回调该方法。该方法必须返回一个 View，该 View 也就是该 Fragment 所显示的 View。
- onPause()：当用户离开该 Fragment 时将会回调该方法。

对于大部分 Fragment 而言，通常都会重写上面这三个方法。但是实际上开发者可以根据需要重写 Fragment 的任意回调方法，后面将会详细介绍 Fragment 的生命周期及其回调方法。

为了控制 Fragment 显示的组件，通常需要重写 onCreateView()方法，该方法返回的 View 将作为该 Fragment 显示的 View 组件。当 Fragment 绘制界面组件时将会回调该方法。

例如如下方法片段：

```
// 重写该方法，该方法返回的 View 将作为 Fragment 显示的组件
@Override
public View onCreateView(LayoutInflater inflater
    , ViewGroup container, Bundle savedInstanceState)
{
    // 加载/res/layout/目录下的 fragment_book_detail.xml 布局文件
    View rootView = inflater.inflate(R.layout.fragment_book_detail,
            container, false);
    if (book != null)
    {
        // 让 book_title 文本框显示 book 对象的 title 属性
        ((TextView) rootView.findViewById(R.id.book_title))
                .setText(book.title);
        // 让 book_desc 文本框显示 book 对象的 desc 属性
        ((TextView) rootView.findViewById(R.id.book_desc))
                .setText(book.desc);
    }
    return rootView;
}
```

上面的方法中第一行粗体字代码使用 LayoutInflater 加载了/res/layout/目录下的 fragment_book_detail.xml 布局文件，最后一行粗体字代码返回该布局文件对应的 View 组件，这表明该 Fragment 将会显示该 View 组件。

实例：开发显示图书详情的 Fragment

下面 Fragment 将会显示加载一份简单的界面布局文件，并根据传入的参数来更新界面组件。该 Fragment 的代码如下。

程序清单：codes\04\4.4\FragmentTest\src\org\crazyit\app\BookDetailFragment.java

```
public class BookDetailFragment extends Fragment
{
    public static final String ITEM_ID = "item_id";
    // 保存该 Fragment 显示的 Book 对象
    BookContent.Book book;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 如果启动该 Fragment 时包含了 ITEM_ID 参数
        if (getArguments().containsKey(ITEM_ID))
        {
            book = BookContent.ITEM_MAP.get(getArguments()
```

```
                .getInt(ITEM_ID)); //①
        }
    }
    // 重写该方法，该方法返回的 View 将作为 Fragment 显示的组件
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        // 加载/res/layout/目录下的 fragment_book_detail.xml 布局文件
        View rootView = inflater.inflate(R.layout.fragment_book_detail,
                container, false);
        if (book != null)
        {
            // 让 book_title 文本框显示 book 对象的 title 属性
            ((TextView) rootView.findViewById(R.id.book_title))
                    .setText(book.title);
            // 让 book_desc 文本框显示 book 对象的 desc 属性
            ((TextView) rootView.findViewById(R.id.book_desc))
                .setText(book.desc);
        }
        return rootView;
    }
}
```

上面的 Fragment 将会加载并显示 res/layout/目录下的 fragment_book_detail.xml 界面布局文件。上面①号代码获取启动该 Fragment 时传入的 ITEM_ID 参数，并根据该 ID 获取 BookContent 的 ITEM_MAP 中的图书信息。

BookContent 类用于模拟系统的数据模型，该模拟类的代码如下。

程序清单：codes\04\4.4\FragmentTest\src\org\crazyit\app\model\BookContent.java

```
public class BookContent
{
    // 定义一个内部类，作为系统的业务对象
    public static class Book
    {
        public Integer id;
        public String title;
        public String desc;
        public Book(Integer id, String title, String desc)
        {
            this.id = id;
            this.title = title;
            this.desc = desc;
        }
        @Override
        public String toString()
        {
            return title;
        }
    }
    // 使用 List 集合记录系统所包含的 Book 对象
    public static List<Book> ITEMS = new ArrayList<Book>();
    // 使用 Map 集合记录系统所包含的 Book 对象
    public static Map<Integer, Book> ITEM_MAP
        = new HashMap<Integer, Book>();
    static
    {
        // 使用静态初始化代码，将 Book 对象添加到 List 集合、Map 集合中
        addItem(new Book(1, "疯狂 Java 讲义"
            , "一本全面、深入的 Java 学习图书，已被多家高校选做教材。"));
```

```
            addItem(new Book(2, "疯狂 Android 讲义"
                , "Android 学习者的首选图书，常年占据京东、当当、 "
                + "亚马逊 3 大网站 Android 销量排行榜的榜首"));
            addItem(new Book(3, "轻量级 Java EE 企业应用实战"
                , "全面介绍 Java EE 开发的 Struts 2、Spring 3、Hibernate 4 框架"));
        }
        private static void addItem(Book book)
        {
            ITEMS.add(book);
            ITEM_MAP.put(book.id, book);
        }
}
```

BookDetailFragment 只是加载并显示一份简单的布局文件，这份布局文件中通过 LinearLayout 包含两个文本框。该布局文件的代码如下。

程序清单：codes\04\4.4\FragmentTest\res\layout\fragment_book_detail.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
<!-- 定义一个 TextView 来显示图书标题 -->
<TextView
    style="?android:attr/textAppearanceLarge"
    android:id="@+id/book_title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp"/>
<!-- 定义一个 TextView 来显示图书描述 -->
<TextView
    style="?android:attr/textAppearanceMedium"
    android:id="@+id/book_desc"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"/>
</LinearLayout>
```

实例：创建 ListFragment

如果开发 ListFragment 的子类，则无须重写 onCreateView()方法——与 ListActivity 类似的是，只要调用 ListFragment 的 setAdapter()方法为该 Fragment 设置 Adapter 即可。该 ListFragment 将会显示该 Adapter 提供的列表项。

下面的实例开发了一个 ListFragment 的子类。

程序清单：codes\04\4.4\FragmentTest\src\org\crazyit\app\BookListFragment.java

```
public class BookListFragment extends ListFragment
{
    private Callbacks mCallbacks;
    // 定义一个回调接口，该 Fragment 所在 Activity 需要实现该接口
    // 该 Fragment 将通过该接口与它所在的 Activity 交互
    public interface Callbacks
    {
        public void onItemSelected(Integer id);
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        // 为该 ListFragment 设置 Adapter
        setListAdapter(new ArrayAdapter<BookContent.Book>(getActivity(),
            android.R.layout.simple_list_item_activated_1,
            android.R.id.text1, BookContent.ITEMS)); //①
    }
    // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 如果 Activity 没有实现 Callbacks 接口，抛出异常
        if (!(activity instanceof Callbacks))
        {
            throw new IllegalStateException(
                "BookListFragment 所在的 Activity 必须实现 Callbacks 接口!");
        }
        // 把该 Activity 当成 Callbacks 对象
        mCallbacks = (Callbacks)activity;
    }
    // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
    @Override
    public void onDetach()
    {
        super.onDetach();
        // 将 mCallbacks 赋为 null。
        mCallbacks = null;
    }
    // 当用户单击某列表项时激发该回调方法
    @Override
    public void onListItemClick(ListView listView
        , View view, int position, long id)
    {
        super.onListItemClick(listView, view, position, id);
        // 激发 mCallbacks 的 onItemSelected 方法
        mCallbacks.onItemSelected(BookContent
            .ITEMS.get(position).id);
    }
    public void setActivateOnItemClick(boolean activateOnItemClick)
    {
        getListView().setChoiceMode(
            activateOnItemClick ? ListView.CHOICE_MODE_SINGLE
                : ListView.CHOICE_MODE_NONE);
    }
}
```

为了控制 ListFragment 显示的列表项，只要调用 ListFragment 提供的 setAdapter()方法即可，这样即可让该 ListFragment 显示该 Adapter 所提供的多个列表项。

4.4.3 Fragment 与 Activity 通信

为了在 Activity 中显示 Fragment，还必须将 Fragment 添加到 Activity 中。将 Fragment 添加到 Activity 中有如下两种方式：

- 在布局文件中使用<fragment.../>元素添加 Fragment，<fragment.../>元素的 android:name 属性指定 Fragment 的实现类。
- 在 Java 代码中通过 FragmentTransaction 对象的 add()方法来添加 Fragment。



提示：

Activity 的 getFragmentManager()方法可返回 FragmentManager，FragmentManager 对象的 beginTransaction()方法即可开启并返回 FragmentTransaction 对象。

下面 Activity 首先通过如下布局文件来使用前面定义的 BookListFragment。

程序清单：codes\04\4.4\FragmentTest\res\layout\activity_book_twopane.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个水平排列的 LinearLayout，并指定使用中等分隔条 -->
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_marginLeft="16dp"
      android:layout_marginRight="16dp"
      android:divider="?android:attr/dividerHorizontal"
      android:showDividers="middle">
      <!-- 添加一个 Fragment -->
      <fragment
            android:name="org.crazyit.app.BookListFragment"
            android:id="@+id/book_list"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1" />
      <!-- 添加一个 FrameLayout 容器 -->
      <FrameLayout
            android:id="@+id/book_detail_container"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="3" />
</LinearLayout>
```

上面的布局文件中粗体字代码使用<fragment.../>元素添加了 BookListFragment，该 Activity 的左边将会显示一个 ListFragment，右边只是一个 FrameLayout 容器，该 FrameLayout 容器将会动态更新其中显示的 Fragment。下面是该 Activity 的代码。

程序清单：codes\04\4.4\FragmentTest\src\org\crazyit\app\SelectBookActivity.java

```
public class SelectBookActivity extends Activity implements
            BookListFragment.Callbacks
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            // 加载/res/layout 目录下的 activity_book_twopane.xml 布局文件
            setContentView(R.layout.activity_book_twopane);
      }
      // 实现 Callbacks 接口必须实现的方法
      @Override
      public void onItemSelected(Integer id)
      {
            // 创建 Bundle，准备向 Fragment 传入参数
            Bundle arguments = new Bundle();
            arguments.putInt(BookDetailFragment.ITEM_ID, id);
            // 创建 BookDetailFragment 对象
            BookDetailFragment fragment = new BookDetailFragment();
            // 向 Fragment 传入参数
```

```
            fragment.setArguments(arguments);
            // 使用 fragment 替换 book_detail_container 容器当前显示的 Fragment
            getFragmentManager().beginTransaction()
                .replace(R.id.book_detail_container, fragment)
                .commit();  //①
        }
    }
```

上面的程序中①号粗体字代码就调用了 FragmentTransaction 的 replace()方法动态更新了 ID 为 book_detail_container 容器（也就是前面布局文件中的 FrameLayout 容器）中显示的 Fragment。

将 Fragment 添加到 Activity 之后，Fragment 必须与 Activity 交互信息，这就需要 Fragment 能获取它所在的 Activity，Activity 也能获取它所包含的任意的 Fragment。可按如下方法进行。

- Fragment 获取它所在的 Activity：调用 Fragment 的 getActivity()方法即可返回它所在的 Activity。
- Activity 获取它包含的 Fragment：调用 Activity 关联的 FragmentManager 的 findFragmentById(int id)或 findFragmentByTag(String tag)方法即可获取指定的 Fragment。



提示：

在界面布局文件中使用<fragment.../>元素添加 Fragment 时，可以为<fragment.../>元素指定 android:id 或 android:tag 属性，这两个属性都可用于标识该 Fragment，接下来 Activity 将可通过 findFragmentById(int id)或 findFragmentByTag (String tag)来获取该 Fragment。

除此之外，Fragment 与 Activity 可能还需要相互传递数据，可按如下方式进行。

- Activity 向 Fragment 传递数据：在 Activity 中创建 Bundle 数据包，并调用 Fragment 的 setArguments(Bundle bundle)方法即可将 Bundle 数据包传给 Fragment。
- Fragment 向 Activity 传递数据或 Activity 需要在 Fragment 运行中进行实时通信：在 Fragment 中定义一个内部回调接口，再让包含该 Fragment 的 Activity 实现该回调接口，这样 Fragment 即可调用该回调方法将数据传给 Activity。

上面实例定义了两个 Fragment，并使用一个 Activity 来“组合”这两个 Activity，该实例的运行界面正是实现图 4.25 中示意界面的左边部分。使用大屏幕设备运行该程序，可以看到如图 4.27 所示界面。



图 4.27　组合 Fragment 实现 UI 界面

4.4.4　Fragment 管理与 Fragment 事务

前面介绍了 Activity 与 Fragment 交互相关的内容，其实 Activity 管理 Fragment 主要依靠

FragmentManger。

FragmentManger 可以完成如下几方面的功能。

- 使用 findFragmentById()或 findFragmentByTag()方法来获取指定 Fragment。
- 调用 popBackStack()方法将 Fragment 从后台栈中弹出（模拟用户按下 BACK 按键）。
- 调用 addOnBackStackChangeListener()注册一个监听器，用于监听后台栈的变化。

如果需要添加、删除、替换 Fragment，则需要借助于 FragmentTransaction 对象，FragmentTransaction 代表 Activity 对 Fragment 执行的多个改变。



提示：

FragmentTransaction 也被翻译为 Fragment 事务。与数据库事务类似的是，数据库事务代表了对底层数组的多个更新操作；而 Fragment 事务则代表了 Activity 对 Fragment 执行的多个改变操作。

开发者可通过 FragmentManager 来获得 FragmentTransaction，代码片段如下：

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

每个 FragmentTransaction 可以包含多个对 Fragment 修改，比如包含调用了多个 add()、remove()、和 replace()操作，最后还调用 commit()方法提交事务即可。

在调用 commit()之前，开发者也可调用 addToBackStack()将事务添加到 back 栈，该栈由 Activity 负责管理，这样允许用户按 BACK 按键返回到前一个 Fragment 状态。

```
// 创建一个新的 Fragment 并打开事务
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();
// 替换该界面中 fragment_container 容器内的 Fragment
transaction.replace(R.id.fragment_container, newFragment);
// 将事务添加到 back 栈，允许用户按 BACK 按键返回到替换 Fragment 之前的状态
transaction.addToBackStack(null);
// 提交事务
transaction.commit();
```

在上面的示例代码中，newFragment 替换了当前界面布局中 ID 为 fragment_container 的容器内的 Fragment，由于程序调用了 addToBackStack()将该 replace 操作添加到了 back 栈中，因此用户可以通过按下 BACK 按键返回替换之前的状态。

前面介绍的实例在大屏幕的平板电脑上运行良好，但在小屏幕的手机上运行就会有问题：运行界面非常丑陋。图 4.28 显示了在小屏幕手机上的运行效果。



图 4.28 小屏幕手机上运行组合 Fragment 实现的 UI 界面

如果希望开发的应用既可在大屏幕平板电脑上使用，也可在小屏幕手机上使用，可以考虑开发兼顾屏幕分辨率的应用。

实例：开发兼顾屏幕分辨率的应用

为了开发兼顾屏幕分辨率的应用，可以考虑在 res/目录下为大屏幕、600dpi 的屏幕建立

相应的资源文件夹：values-large，values-sw600dp，在该文件夹下建立一个名为 refs.xml 的引用资源文件。该引用资源文件专门用于定义各种引用项。

本实例的引用资源文件中只有一项，下面是该引用资源文件的代码。

程序清单：codes\04\4.4\SeniorFragmentTest\res\values-large\refs.xml

```
<resources>
    <!-- 定义 activity_book_list 实际引用@layout/activity_book_twopane 资源 -->
    <item type="layout" name="activity_book_list">
        @layout/activity_book_twopane</item>
</resources>
```

上面引用资源文件中指定 activity_book_list 引用/res/layout/目录下的 activity_book_twopane.xml 界面布局文件。

接下来在 Activity 加载 R.layout.activity_book_list 时将会根据运行平台的屏幕大小自动选择界面布局文件：在大屏幕的平板电脑上，R.layout.activity_book_list 将会变成/res/layout/目录下的 activity_book_twopane 界面布局文件；在小屏幕的手机上，R.layout.activity_book_list 依然引用/res/layout/目录下的 activity_book_list.xml 界面布局文件。

下面是 activity_book_list.xml 界面布局文件的代码。

程序清单：codes\04\4.4\SeniorFragmentTest\res\layout\activity_book_list.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 直接使用 BookListFragment 作为界面组件 -->
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:name="org.crazyit.app.BookListFragment"
    android:id="@+id/book_list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"/>
```

从上面的布局文件可以看出，该布局文件仅仅是显示 BookListFragment 组件，这表明该界面布局文件中只是显示图书列表。

接下来加载该布局文件的 Activity 将会针对不同屏幕分辨率分别进行处理。下面是该 Activity 的代码。

程序清单：codes\04\4.4\SeniorFragmentTest\src\org\crazyit\app\BookListActivity.java

```
public class BookListActivity extends Activity implements
        BookListFragment.Callbacks
{
    // 定义一个旗标，用于标识该应用是否支持大屏幕
    private boolean mTwoPane;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 指定加载 R.layout.activity_book_list 对应的界面布局文件
        // 但实际上该应用会根据屏幕分辨率加载不同的界面布局文件
        setContentView(R.layout.activity_book_list);
        // 如果加载的界面布局文件中包含 ID 为 book_detail_container 的组件
        if (findViewById(R.id.book_detail_container) != null)
        {
            mTwoPane = true;
            ((BookListFragment) getFragmentManager()
                .findFragmentById(R.id.book_list))
                .setActivateOnItemClick(true);
```

```
            }
        }
        @Override
        public void onItemSelected(Integer id)
        {
            if (mTwoPane)
            {
                // 创建 Bundle，准备向 Fragment 传入参数
                Bundle arguments = new Bundle();
                arguments.putInt(BookDetailFragment.ITEM_ID, id);
                // 创建 BookDetailFragment 对象
                BookDetailFragment fragment = new BookDetailFragment();
                // 向 Fragment 传入参数
                fragment.setArguments(arguments);
                // 使用 fragment 替换 book_detail_container 容器当前显示的 Fragment
                getFragmentManager().beginTransaction()
                        .replace(R.id.book_detail_container, fragment).commit();
            }
            else
            {
                // 创建启动 BookDetailActivity 的 Intent
                Intent detailIntent = new Intent(this, BookDetailActivity.class);
                // 设置传给 BookDetailActivity 的参数
                detailIntent.putExtra(BookDetailFragment.ITEM_ID, id);
                // 启动 Activity
                startActivity(detailIntent);
            }
        }
    }
```

由于该实例为大屏幕设备定义了引用资源文件，因此该应用将会根据大屏幕加载对应的界面布局文件，因此上面程序中第一段粗体字代码会判断界面布局中是否包含 ID 为 book_detail_container 的组件，如果包含该组件则表明是适应大屏幕的“双屏”界面。否则，该程序界面上只包含一个简单的 BookListFragment 列表组件。此时当用户单击列表项时，程序不再是简单地更换 Fragment，而是启动 BookDetailActivity 来显示指定图书的详细信息。

BookDetailActivity 只是一个简单的封装，它将直接复用前面已有的 BookDetailFragment。BookDetailActivity 的代码如下。

程序清单：codes\04\4.4\SeniorFragmentTest\src\org\crazyit\app\BookDetailActivity.java

```
public class BookDetailActivity extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 指定加载/res/layout 目录下的 activity_book_detail.xml 布局文件
        // 该界面布局文件内只定义了一个名为 book_detail_container 的 FrameLayout
        setContentView(R.layout.activity_book_detail);
        // 将 ActionBar 上应用图标转换成可点击的按钮
        getActionBar().setDisplayHomeAsUpEnabled(true);
        if (savedInstanceState == null)
        {
            // 创建 BookDetailFragment 对象
            BookDetailFragment fragment = new BookDetailFragment();
            // 创建 Bundle 对象，
            Bundle arguments = new Bundle();
            arguments.putInt(BookDetailFragment.ITEM_ID, getIntent()
```

```
                    .getIntExtra(BookDetailFragment.ITEM_ID, 0));
                // 向 Fragment 传入参数
                fragment.setArguments(arguments);
                // 将指定 fragment 添加到 book_detail_container 容器中
                getFragmentManager().beginTransaction()
                        .add(R.id.book_detail_container, fragment).commit();
            }
        }
        @Override
        public boolean onOptionsItemSelected(MenuItem item)
        {
            if (item.getItemId() == android.R.id.home)
            {
                // 创建启动 BookListActivity 的 Intent
                Intent intent = new Intent(this, BookListActivity.class);
                // 添加额外的 Flag，将 Activity 栈中处于 FirstActivity 之上的 Activity 弹出
                intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                // 启动 intent 对应的 Activity
                startActivity(intent);
                return true;
            }
            return super.onOptionsItemSelected(item);
        }
    }
```

上面的粗体字代码创建了 BookDetailFragment，并让该 Activity 显示该 Fragment 即可。

除此之外，该 Activity 还启用了 ActionBar 上的应用程序图标，允许用户点击该图标返回程序的主 Acitivity。

在大屏幕设备上运行该实例，可以看到如图 4.29 所示界面。

单击其中一个列表项，将可以看到 4.30 所示界面。



图 4.29　图书列表界面



图 4.30　图书详情界面

4.5　Fragment 的生命周期

与 Activity 类似的是，Fragment 也存在如下状态。

- 活动状态：当前 Fragment 位于前台，用户可见，可以获得焦点。
- 暂停状态：其他 Activity 位于前台，该 Fragment 依然可见，只是不能获得焦点。
- 停止状态：该 Fragment 不可见，失去焦点。
- 销毁状态：该 Fragment 被完全删除，或该 Fragment 所在的 Activity 被结束。



提示：

Adnroid 文档只提到了 Fragment 的三个状态，官方文档没有提到 Fragment 的“销毁状态”。这也是合理的，因为处于“销毁状态”的 Fragment 基本不可用了，只能等着被回收了。

图4.31（该图参照Android官方文档，有些地方与文档有差异，以实际运行结果为准确）显示了Fragment生命周期及相关回调方法。

从图4.31可以看出，在Fragment的生命周期中，如下方法会被系统回调。

- onAttach()：当该Fragment被添加到Activity时被回调。该方法只会被调用一次。
- onCreate(Bundle savedStatus)：创建Fragment时被回调。该方法只会被调用一次。
- onCreateView()：每次创建、绘制该Fragment的View组件时回调该方法，Fragment将会显示该方法返回的View组件。
- onActivityCreated()：当Fragment所在的Activity被启动完成后回调该方法。
- onStart()：启动Fragment时被回调。
- onResume()：恢复Fragment时被回调，onStart()方法后一定会回调onResume()方法。
- onPause()：暂停Fragment时被回调。
- onStop()：停止Fragment时被回调。
- onDestroyView()：销毁该Fragment所包含的View组件时调用。
- onDestroy()：销毁Fragment时被回调。该方法只会被调用一次。
- onDetach()：将该Fragment从Activity中被删除、被替换完成时回调该方法，onDestroy()方法后一定会回调onDetach()方法。该方法只会被调用一次。

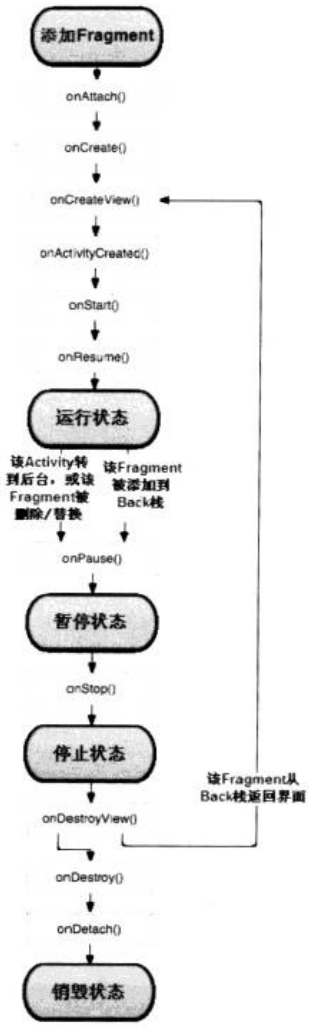


图4.31 Fragment的生命周期及其回调方法

正如开发Activity时可以根据需要选择性地覆盖指定方法，开发Fragment时也可根据需要选择性地覆盖指定方法。其中最常见的就是覆盖onCreateView()方法——该方法返回的View将由Fragment显示出来。

下面的Fragment覆盖了上面的11个生命周期方法，并在每个方法中增加了一行记录日志代码。该Fragment的代码如下。

程序清单：codes\04\4.5\FragmentLifecycle\src\org\crazyit\app\LifecycleFragment.java

```
public class LifecycleFragment extends Fragment
{
    final String TAG = "--CrazyIt--";
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 输出日志
        Log.d(TAG, "-------onAttach------");
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        // 输出日志
        Log.d(TAG, "-------onCreate------");
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle data)
    {
        // 输出日志
        Log.d(TAG, "-------onCreateView------");
        TextView tv = new TextView(getActivity());
        tv.setGravity(Gravity.CENTER_HORIZONTAL);
        tv.setText("测试 Fragment");
        tv.setTextSize(40);
        return tv;
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState)
    {
        super.onActivityCreated(savedInstanceState);
        // 输出日志
        Log.d(TAG, "-------onActivityCreated------");
    }
    @Override
    public void onStart()
    {
        super.onStart();
        // 输出日志
        Log.d(TAG, "-------onStart------");
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 输出日志
        Log.d(TAG, "-------onResume------");
    }
    @Override
    public void onPause()
    {
        super.onPause();
        // 输出日志
        Log.d(TAG, "-------onPause------");
    }
    @Override
    public void onStop()
    {
        super.onStop();
        // 输出日志
        Log.d(TAG, "-------onStop------");
    }
    @Override
    public void onDestroyView()
    {
        super.onDestroyView();
        // 输出日志
        Log.d(TAG, "-------onDestroyView------");
    }
    @Override
    public void onDestroy()
    {
```

```
            super.onDestroy();
            // 输出日志
            Log.d(TAG, "-------onDestroy------");
        }
        @Override
        public void onDetach()
        {
            super.onDetach();
            // 输出日志
            Log.d(TAG, "-------onDetach------");
        }
}
```

包含该 Fragment 的 Activity 的布局文件很简单：该布局文件中只包含一个容器来盛装 Fragment，另外还包含几个按钮。此处不给出界面布局代码。

当使用 Activity 加载该 Fragment 时可以在 LogCat 控制台看到如图 4.32 所示输出。

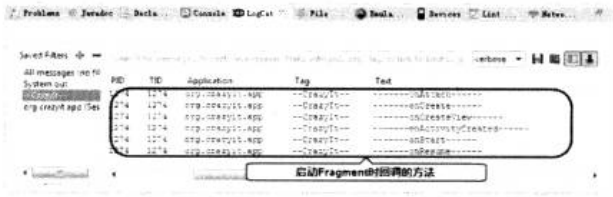


图 4.32 启动 Fragment 时回调的方法

如果单击程序中“启动对话框风格的 Activity”按钮，将启动一个对话框风格的 Activity，当前 Activity 将会转入“暂停”状态，该 Fragment 已经进入“暂停”状态，此时可以在 LogCat 看到如图 4.33 所示输出。



图 4.33 暂停 Fragment 时回调的方法

关闭对话框风格的 Activity，Fragment 将会再次进入“运行”状态，将可以在 LogCat 看到如图 4.34 所示的输出。



图 4.34 恢复 Fragment 时回调的方法

单击程序界面上“替换目标 Fragment、并加入 Back 栈”，将可以在 LogCat 看到如图 4.35 所示的输出。

从图 4.35 可以看出，替换目标 Fragment、并将它添加到 Back 栈中，此时该 Fragment 虽然并不可见，但它并未被销毁，它只是被添加到后台的 Back 栈中。当用户按下手机的键时，该 Fragment 将会再次显示出来，此时可以在 LogCat 看到如图 4.36 所示的输出。

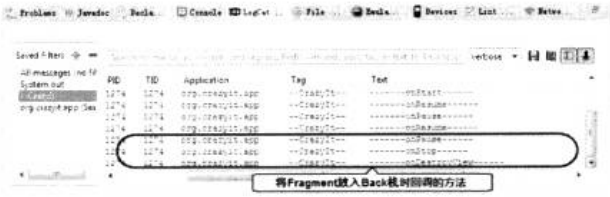


图 4.35　将 Fragment 加入 Back 栈时回调的方法



图 4.36　将 Back 栈中 Fragment 转入前台时回调的方法

单击界面上“替换目标 Fragment”或“退出”按钮，该 Fragment 将会被完全结束，Fragent 将进入“销毁”状态，此时可以在 LogCat 看到如图 4.37 所示输出。



图 4.37　结束 Fragment 时回调的方法

4.6　本章小结

本章详细介绍了 Android 四大组件之一：Activity。一个 Android 应用将会包含多个 Activity，每个 Activity 通常对应于一个窗口。普通用户接触最多的就是 Activity。学习本章的重点是掌握如何开发 Activity、如何在 AndroidManifest.xml 文件中配置 Activity。不仅如此，由于 Android 系统通常会由多个 Activity 组成，因此读者还需要掌握启动其他 Activity 的方法，包括如何利用 Bundle 在不同 Activity 之间通信，启动其他 Activity 并返回结果等。Activity 在 Android 系统中运行，具有自身的生命周期，因此读者还需要清晰地掌握 Activity 的生命周期。

本章的另一个重点是掌握开发 Fragment 的方法，包括为 Fragment 开发界面、把 Fragment 添加到 Activity、Fragment 与 Activity 通信等内容。Fragment 也具有自己的生命周期，因此读者也需要清晰地掌握 Fragment 的生命周期。

CHAPTER

5

第5章 使用 Intent 和 IntentFilter 进行通信

本章要点

- 理解 Intent 对于 Android 应用的作用
- 使用 Intent 启动系统组件
- Intent 的 Component 属性的作用
- Intent 的 Action 属性的作用
- Intent 的 Category 属性的作用
- 为指定 Action、Category 的 Intent 配置对应的 intent-filter
- Intent 的 Data 属性
- Intent 的 Type 属性
- 为指定 Data、Type 的 Intent 配置对应的 intent-filter
- Intent 的 Extra 属性
- Intent 的 Flag 属性
- 使用 Intent 创建 Tab 页

前面介绍 Activity 时已经多次使用了 Intent，当一个 Activity 需要启动另一个 Activity 时，程序并没有直接告诉系统要启动哪个 Activity，而是通过 Intent 来表达自己的意图：需要启动哪个 Activity。“Intent”的中文翻译就是“意图”的意思。

在这里有读者可能会产生一个疑问，假如甲 Activity 需要启动另一个 Activity，为何不直接使用一个形如 startActivity(Class activityClass)的方法呢？这样多么简单、明了啊。但实际上，这种方式虽然简洁，却明显背离了 Android 的理念，Android 使用 Intent 来封装程序的“调用意图”，不管程序想启动一个 Activity 也好，想启动一个 Service 组件也好，想启动一个 BroadcastReceiver 也好，Android 使用统一的 Intent 对象来封装这种“启动意图”，很明显使用 Intent 提供了一致的编程模型。

除此之外，使用 Intent 还有一个好处：在某些时候，应用程序只是想启动具有某种特征的组件，并不想和某个具体的组件耦合，如果使用形如 startActivity(Class activityClass)的方法来启动特定组件，势必造成一种硬编码耦合，这样也不利于高层次的解耦。



提示：

如果读者有过 Struts 2 等 MVC 框架的编程经验，一定可以很好地理解此处 Intent 的设计，当 Struts 2 的 Action 处理完用户请求之后，它并不会直接返回特定的视图页面，而是返回一个逻辑视图名，开发者可以在配置文件中把该逻辑视图映射到任意的物理视图资源；Android 系统的 Intent 设计有点类似于 Struts 2 框架中逻辑视图的设计。

总之，Intent 封装 Android 应用程序需要启动某个组件的“意图”。不仅如此，Intent 还是应用程序组件之间通信的重要媒介，正如前面程序看到的，两个 Activity 可以把需要交换的数据封装成 Bundle 对象，然后使用 Intent 来携带 Bundle 对象，这样就实现了两个 Activity 之间的数据交换。

5.1 Intent 对象详解

前面介绍 Activity 时已经多次使用了 Intent，相信读者对 Intent 对象已经有了一个初步的认识，下面将对 Intent 对象进行更全面的介绍。

➤➤5.1.1 使用 Intent 启动系统组件

Android 的应用程序包含三种重要组件：Activity、Service、BroadcastReceiver，应用程序采用了一致的方式来启动它们——都是依靠 Intent 来进行启动的，Intent 就封装了程序想要启动程序的意图，不仅如此，Intent 还可用于与被启动组件交换信息。

表 5.1 显示使用 Intent 启动不同组件的方法。

表 5.1　使用 Intent 启动不同组件的方法

组件类型	启动方法
Activity	startActivity(Intent intent) startActivityForResult(Intent intent, int requestCode)

续表

组件类型	启动方法
Service	ComponentName startService(Intent service) boolean bindService(Intent service, ServiceConnection conn, int flags)
BroadcastReceiver	sendBroadcast(Intent intent) sendBroadcast(Intent intent, String receiverPermission) sendOrderedBroadcast(Intent intent, String receiverPermission, BroadcastReceiver resultReceiver, Handler scheduler, int initialCode, String initialData, Bundle initialExtras) sendOrderedBroadcast(Intent intent, String receiverPermission) sendStickyBroadcast(Intent intent) sendStickyOrderedBroadcast(Intent intent, BroadcastReceiver resultReceiver, Handler scheduler, int initialCode, String initialData, Bundle initialExtras)

上一章我们已经见到了如何使用 Intent 来启动 Activity 的示例，至于使用 Intent 来启动另外两种组件的示例，本书后面章节会有相关示例，此处暂不深入介绍。

Intent 对象大致包含 Component、Action、Category、Data、Type、Extra 和 Flag 这 7 种属性，其中 Component 用于明确指定需要启动的目标组件，而 Extra 则用于“携带”需要交换的数据。

下面详细介绍 Intent 对象各属性的作用。

5.2 Intent 的属性及 intent-filter 配置

Intent 代表了 Android 应用的启动“意图”，Android 应用将会根据 Intent 来启动指定组件，至于到底启动哪个组件，则取决于 Intent 的各属性。下面将详细介绍 Intent 的各属性值，以及 Android 如何根据不同属性值来启动相应的组件。

5.2.1 Component 属性

Intent 的 Component 属性需要接受一个 ComponentName 对象，ComponentName 对象包含如下几个构造器。

- ComponentName(String pkg, String cls)：创建 pkg 所在包下的 cls 类所对应的组件。
- ComponentName(Context pkg, String cls)：创建 pkg 所对应的包下的 cls 类所对应的组件。
- ComponentName(Context pkg, Class<?> cls)：创建 pkg 所对应的包下的 cls 类所对应的组件。

上面构造器的本质就是一个，这说明创建一个 ComponentName 需要指定包名和类名——这就可唯一地确定一个组件类，这样应用程序即可根据给定的组件类去启动特定的组件。

除此之外，Intent 还包含了如下三个方法。

- setClass(Context packageContext, Class<?> cls)：设置该 Intent 将要启动的组件对应的类。
- setClassName(Context packageContext, String className)：设置该 Intent 将要启动的组件对应的类名。

- setClassName(String packageName, String className)：设置该 Intent 将要启动的组件对应的类名。



提示：

Android 应用的 Context 代表了访问该应用环境信息的接口，而 Android 应用的包名则作为应用的唯一标识，因此 Android 应用的 Context 对象与该应用的包名有一一对应的关系。上面三个 setClass()方法就是指定了包名（分别通过 Context 指定或 String 指定）和组件的实现类（分别通过 Class 指定或通过 String 指定）。

指定了 Component 属性的 Intent 已经明确了它将要启动哪个组件，因此这种 Intent 也被称为显式 Intent，没有指定 Component 属性的 Intent 被称为隐式 Intent——隐式 Intent 没有明确指定要启动哪个组件，应用将会根据 Intent 指定的规则去启动符合条件的组件，但具体是哪个组件则不确定。



提示：

比如一个女孩子有找男朋友的意图，此时有两种方式来表达她的意图：第一种，她明确指出，要找“梁山伯”做男朋友，这种明确指定要找某人的方式被称为显式 Intent；第二种：她指出，要找“高”、“富”、“帅”做男朋友，至于到底是谁不重要，只要符合这三个条件即可，这种方式被称为隐式 Intent。

下面的示例程序示范了如何通过显式 Intent（指定了 Component 属性）来启动另一个 Activity。该程序的界面布局很简单，界面中只有一个按钮，用户单击该按钮将会启动第二个 Activity。此处不再给出该程序的界面布局文件。该程序的 Java 代码如下。

程序清单：codes\05\5.2\ComponentAttr\src\org\crazyit\intent\ComponentAttr.java

```
public class ComponentAttr extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 创建一个 ComponentName 对象
                ComponentName comp = new ComponentName(ComponentAttr.this,
                        SecondActivity.class);
                Intent intent = new Intent();
                // 为 Intent 设置 Component 属性
                intent.setComponent(comp);
                startActivity(intent);
            }
        });
    }
}
```

上面的程序中三行粗体字代码用于创建 ComponentName 对象，并将该对象设置成 Intent

对象的 Component 属性，这样应用程序即可根据该 Intent 的“意图”去启动指定组件。

实际上，上面三行粗体字代码完全可以简化为如下形式：

```
// 根据指定组件类来创建 Intent
Intent intent = new Intent(ComponentAttr.this
      , SecondActivity.class);
```

从上面的代码可以看出，当需要为 Intent 设置 Component 属性时，实际上 Intent 已经提供了一个简化的构造器，这样方便程序直接指定启动其他组件。

当程序通过 Intent 的 Component 属性（明确指定了启动哪个组件）启动特定组件时，被启动组件几乎不需要使用<intent-filter.../>元素进行配置。

程序的 SecondActivity 也很简单，它的界面布局中只包含一个简单的文本框，用于显示该 Activity 对应的 Intent 的 Component 属性的包名、类名，该 Activity 的 Java 代码如下。

程序清单：codes\05\5.2\ComponentAttr\src\org\crazyit\intent\SecondActivity.java

```
public class SecondActivity extends Activity
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.second);
            EditText show = (EditText) findViewById(R.id.show);
            // 获取该 Activity 对应的 Intent 的 Component 属性
            ComponentName comp = getIntent().getComponent();
            // 显示该 ComponentName 对象的包名、类名
            show.setText("组件包名为：" + comp.getPackageName()
                  + "\n 组件类名为：" + comp.getClassName());
      }
}
```

运行上面的程序，通过第一个 Activity 中的按钮进入第二个 Activity 将可以看到如图 5.1 所示的界面。

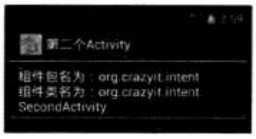


图 5.1 Intent 的 Component 属性

5.2.2 Action、Category 属性与 intent-filter 配置

Intent 的 Action、Category 属性都是一个普通的字符串，其中 Action 代表该 Intent 所要完成的一个抽象“动作”，而 Category 则用于为 Action 增加额外的附加类别信息。通常 Action 属性会与 Category 属性结合使用。

Action 要完成的只是一个抽象的动作，这个动作具体由哪个组件（或许是 Activity，或者是 BroadcastReceiver）来完成，Action 这个字符串本身并不管。比如 Android 提供的标准 Action：Intent. ACTION_VIEW，它只表示一个抽象的查看操作，但具体查看什么、启动哪个 Activity 来查看，Intent. ACTION_VIEW 并不知道——这取决于 Activity 的<intent-filter.../>配置，只要某个 Activity 的<intent-filter.../>配置中包含了该 ACTION_VIEW，该 Actvitiy 就有可能被启动。



提示：

有过 Struts 2 开发经验的读者都知道，当 Struts 2 的 Action 处理用户请求结束后，Action 并不会直接指定“跳转”到哪个 Servlet（通常是 JSP，但 JSP 的本质就是 Servlet），Action 的处理方法只是返回一个普通字符串，然后在配置文件中配置该字符串对应到哪个 Servlet。Struts 2 之所以采用这种设计思路，就是为了把 Action 与呈现视图的 Servlet 分离开。类似的，Intent 通过指定 Action 属性（其实就是一个普通字符串），就可以把该 Intent 与具体的 Activity 分离，从而提供高层次的解耦。

下面通过一个简单的示例来示范 Action 属性（就是普通字符串）的作用。下面的程序的第一个 Activity 非常简单，它只包括一个普通按钮，当用户单击该按钮时，程序会“跳转”到第二个 Activity——但第一个 Activity 指定跳转的 Intent 时，并不以“硬编码”的方式指定要跳转的目标 Activity，而是为 Intent 指定 Action 属性。此处不给出界面布局的代码，第一个 Activity 的 Java 代码如下。

程序清单：codes\05\5.2\ActionAttr\src\org\crazyit\intent\ActionAttr.java

```
public class ActionAttr extends Activity
{
    public final static String CRAZYIT_ACTION =
        "org.crazyit.intent.action.CRAZYIT_ACTION";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                // 为 Intent 设置 Action 属性（属性值就是一个普通字符串）
                intent.setAction(ActionAttr.CRAZYIT_ACTION);
                startActivity(intent);
            }
        });
    }
}
```

上面的程序中粗体字代码指定了根据 Intent 来启动 Activity——但该 Intent 并未以“硬编码”的方式指定要启动哪个 Activity，相信读者从上面程序的粗体字代码无法看出该程序将要启动哪个 Activity。那么到底程序会启动哪个 Activity 呢？这取决于 Activity 配置中<intent-filter.../>元素的配置。

<intent-filter.../>元素是 AndroidManifest.xml 文件中<activty.../>元素的子元素，前面已经介绍过，<activity.../>元素用于为应用程序配置 Activity，<activity.../>的<intent-filter.../>子元素则用于配置该 Activity 所能“响应”的 Intent。

<intent-filter.../>元素里通常可包含如下子元素。

- 0～*N* 个<action.../>子元素。
- 0～*N* 个<category.../>子元素。
- 0～1 个<data.../>子元素。



提示：

<intent-filter.../>元素也可是<service.../>、<receiver.../>两个元素的子元素，用于表明它们可以响应的 Intent。

<action.../>、<category.../>子元素的配置非常简单，它们都可指定 android:name 属性，该属性的值就是一个普通字符串。

当<activity.../>元素里的<intent-filter.../>子元素里包含多个<action.../>子元素（相当于指定了多个字符串）时，就表明该 Activity 能响应 Action 属性值为其中任意一个字符串的 Intent。



提示：

还是借用前面介绍的女孩子找男朋友的例子，基本上可以把 Intent 中的 Action 属性、Category 属性，理解为她对男朋友的要求，每个 Intent 只能指定一个 Action “要求”，但可以指定多个 Category 要求；IntentFilter（使用<intent-filter.../>元素配置）则用于声明该组件（比如 Activity、Service、BroadcastReceiver）能满足的要求，每个组件可以声明自己满足多个 Action 要求、可满足多个 Category 要求。只要某个组件能满足的要求大于、等于 Intent 所指定的要求，那么该 Intent 就能启动该组件。

由于上面的程序指定启动 Action 属性为 ActionAttr.CRAZYIT_ACTION 常量（常量值为 org.crazyit.intent.action.CRAZYIT_ACTION）的 Activity，也就要求被启动 Activity 对应的配置元素的<intent-filter.../>元素里至少包括一个如下的<action.../>子元素：

```
<action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
```

需要指出的是，一个 Intent 对象最多只能包括一个 Action 属性，程序可调用 Intent 的 setAction(String str)方法来设置 Action 属性值；但一个 Intent 对象可以包含多个 Category 属性，程序可调用 Intent 的 addCategory (String str)方法来为 Intent 添加 Category 属性。当程序创建 Intent 时，该 Intent 默认启动 Category 属性值为 Intent.CATEGORY_DEFAULT 常量（常量值为 android.intent.category.DEFAULT）的组件。

因此，虽然上面程序的粗体字代码并未指定目标的 Intent 的 Category 属性，但该 Intent 已有一个值为 android.intent.category.DEFAULT 的 Category 属性值，因此被启动 Activity 对应的配置元素的<intent-filter.../>元素里至少还包括一个如下的<category.../>子元素：

```
<category android:name="android.intent.category.DEFAULT" />
```

下面是被启动的 Activity 的完整配置。

程序清单：codes\05\5.2\ActionAttr\AndroidManifest.xml

```
<activity android:name=".SecondActivity"
          android:label="@string/app_name">
    <intent-filter>
        <!-- 指定该 Activity 能响应 Action 为指定字符串的 Intent -->
        <action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
        <!-- 指定该 Activity 能响应 Action 属性为 helloWorld 的 Intent -->
```

```
            <action android:name="helloWorld" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </activity>
```

上面 Activity 配置的三行粗体字代码指定该 Activity 能具有指定 Action 属性值、指定 Category 属性值的 Intent。其中第二条粗体字代码只是试验用的，对于本程序没有影响——它表明该 Activity 能响应 Action 属性为 helloWorld 字符串、Category 属性值为 android.intent.category.DEFAULT 的 Intent，但我们的程序并未尝试启动这样的 Activity，读者可以自己尝试用这样的 Intent 来启动 Activity，将会看到程序也是启动该 Activity。

上面的配置代码中配置了一个实现类为 SecondActivity 的 Activity，因此程序还应该提供这个 Activity 代码，代码如下。

程序清单：codes\05\5.2\ActionAttr\src\org\crazyit\intent\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        EditText show = (EditText)findViewById(R.id.show);
        // 获取该 Activity 对应的 Intent 的 Action 属性
        String action = getIntent().getAction();
        // 显示 Action 属性
        show.setText("Action 为：" + action);
    }
}
```

上面的程序代码很简单，它只是在启动时把启动该 Activity 的 Intent 的 Action 属性显示在指定文本框内。运行上面的程序并单击程序中“启动指定 Action、默认 Category 对应的 Activity”按钮，将看到如图 5.2 所示界面。



图 5.2　根据 Intent 的 Action 来启动 Activity

接下来的示例程序将会示范 Category 属性的用法，该程序的第一个 Activity 代码如下。

程序清单：codes\05\5.2\ActionCateAttr\src\org\crazyit\intent\ActionCateAttr.java

```
public class ActionCateAttr extends Activity
{
    // 定义一个 Action 常量
    final static String CRAZYIT_ACTION =
        "org.crazyit.intent.action.CRAZYIT_ACTION";
    // 定义一个 Category 常量
    final static String CRAZYIT_CATEGORY =
        "org.crazyit.intent.category.CRAZYIT_CATEGORY";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
```

```
                {
                    Intent intent = new Intent();
                    // 设置Action属性
                    intent.setAction(ActionCateAttr.CRAZYIT_ACTION);
                    // 添加Category属性
                    intent.addCategory(ActionCateAttr.CRAZYIT_CATEGORY);
                    startActivity(intent);
                }
            });
        }
}
```

上面的程序中两行粗体字代码指定了该 Intent 的 Action 属性为 org.crazyit.intent.action.CRAZYIT_ACTION 字符串，并为该 Intent 添加了字符串为 org.crazyit.intent.category.CRAZYIT_CATEGORY 的 Category 属性。这意味着上面的程序所要启动的目标 Activity 里应该包含如下<action.../>子元素和<category.../>子元素：

```
<!-- 指定该Activity能响应action为指定字符串的Intent -->
<action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
<!-- 指定该Activity能响应category为指定字符串的Intent -->
<category android:name="org.crazyit.intent.category.CRAZYIT_CATEGORY" />
<!-- 指定该Activity能响应category为android.intent.category.DEFAULT的Intent -->
<category android:name="android.intent.category.DEFAULT" />
```

下面是程序要启动的目标 Action 所对应的配置代码。

程序清单：codes\05\5.2\ActionCateAttr\AndroidManifest.xml

```
<activity android:name=".SecondActivity"
            android:label="@string/app_name">
    <intent-filter>
        <!-- 指定该Activity能响应action为指定字符串的Intent -->
        <action android:name="org.crazyit.intent.action.CRAZYIT_ACTION" />
        <!-- 指定该Activity能响应category为指定字符串的Intent -->
        <category android:name="org.crazyit.intent.category.CRAZYIT_CATEGORY" />
        <!-- 指定该Activity能响应category为android.intent.category.DEFAULT的
        Intent -->
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

上面配置 Activity 时也指定该 Activity 的实现类为 SecondActivity，该实现类的代码如下。

程序清单：codes\05\5.2\ActionCateAttr\src\org\crazyit\intent\SecondActivity.java

```
public class SecondActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.second);
        EditText show = (EditText) findViewById(R.id.show);
        // 获取该Activity对应的Intent的Action属性
        String action = getIntent().getAction();
        // 显示Action属性
        show.setText("Action为：" + action);
        EditText cate = (EditText) findViewById(R.id.cate);
        // 获取该Activity对应的Intent的Category属性
        Set<String> cates = getIntent().getCategories();
```

```
            // 显示 Action 属性
            cate.setText("Category 属性为：" + cates);
        }
    }
```

上面的程序也很简单，它只是在启动时把启动该 Activity 的 Intent 的 Action、Catetory 属性分别显示在不同的文本框内。运行上面的程序，并单击程序中“启动指定 Action、指定 Category 对应的 Activity”按钮，将看到如图 5.3 所示的界面。

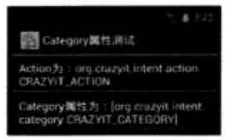


图 5.3　根据 Intent 的 Action、Category 来启动 Activity

5.2.3　指定 Action、Category 调用系统 Activity

Intent 代表了启动某个程序组件的“意图”，实际上 Intent 对象不仅可以启动本应用内程序组件，也可启动 Android 系统的其他应用的程序组件，包括系统自带的程序组件——只要权限允许。

实际上 Android 内部提供了大量标准 Action、Catetory 常量，其中用于启动 Activity 的标准 Action 常量及对应的字符串如表 5.2 所示。

表 5.2　启动 Activity 的标准 Action

Action 常量	对应字符串	简单说明
ACTION_MAIN	android.intent.action.MAIN	应用程序入口
ACTION_VIEW	android.intent.action.VIEW	显示指定数据
ACTION_ATTACH_DATA	android.intent.action.ATTACH_DATA	指定某块数据将被附加到其他地方
ACTION_EDIT	android.intent.action.EDIT	编辑指定数据
ACTION_PICK	android.intent.action.PICK	从列表中选择某项，并返回所选的数据
ACTION_CHOOSER	android.intent.action.CHOOSER	显示一个 Activity 选择器
ACTION_GET_CONTENT	android.intent.action.GET_CONTENT	让用户选择数据，并返回所选数据
ACTION_DIAL	android.intent.action.DIAL	显示拨号面板
ACTION_CALL	android.intent.action.CALL	直接向指定用户打电话
ACTION_SEND	android.intent.action.SEND	向其他人发送数据
ACTION_SENDTO	android.intent.action.SENDTO	向其他人发送消息
ACTION_ANSWER	android.intent.action.ANSWER	应答电话
ACTION_INSERT	android.intent.action.INSERT	插入数据
ACTION_DELETE	android.intent.action.DELETE	删除数据
ACTION_RUN	android.intent.action.RUN	运行数据
ACTION_SYNC	android.intent.action.SYNC	执行数据同步
ACTION_PICK_ACTIVITY	android.intent.action.PICK_ACTIVITY	用于选择 Activity
ACTION_SEARCH	android.intent.action.SEARCH	执行搜索
ACTION_WEB_SEARCH	android.intent.action.WEB_SEARCH	执行 Web 搜索
ACTION_FACTORY_TEST	android.intent.action.FACTORY_TEST	工厂测试的入口点

标准 Category 常量及对应的字符串如表 5.3 所示。

表 5.3 标准 Category

Category 常量	对应字符串	简单说明
CATEGORY_DEFAULT	android.intent.category.DEFAULT	默认的 Category
CATEGORY_BROWSABLE	android.intent.category.BROWSABLE	指定该 Activity 能被浏览器安全调用
CATEGORY_TAB	android.intent.category.TAB	指定 Activity 作为 TabActivity 的 Tab 页
CATEGORY_LAUNCHER	android.intent.category. LAUNCHER	Activity 显示顶级程序列表中
CATEGORY_INFO	android.intent.category. INFO	用于提供包信息
CATEGORY_HOME	android.intent.category. HOME	设置该 Activity 随系统启动而运行
CATEGORY_PREFERENCE	android.intent.category.PREFERENCE	该 Activity 是参数面板
CATEGORY_TEST	android.intent.category.TEST	该 Activity 是一个测试
CATEGORY_CAR_DOCK	android.intent.category.CAR_DOCK	指定手机被插入汽车底座（硬件）时运行该 Activity
CATEGORY_DESK_DOCK	android.intent.category.DESK_DOCK	指定手机被插入桌面底座（硬件）时运行该 Activity
CATEGORY_CAR_MODE	android.intent.category.CAR_MODE	设置该 Activity 可在车载环境下使用



备注： 表 5.2、表 5.3 所列出的都只是部分较为常用的 Action 常量、Category 常量，关于 Intent 所提供的全部 Action 常量、Category 常量，应参考 Android API 文档中关于 Intent 的说明。

下面将以两个实例来介绍 Intent 系统 Action、系统 Category 的用法。

实例：查看并获取联系人电话

这个程序将会在程序中提供一个按钮，用户单击该按钮时会显示系统的联系人列表，当用户单击指定联系人之后，程序将会显示该联系人的名字、电话。



提示： 这个程序非常有用，比如我们要开发一个发送短信的程序，当用户短信编写完成之后，可能需要浏览联系人列表，并从联系人列表中选出短信接收人，那就可以用到该程序了。

该程序的界面布局代码如下。

程序清单：codes\05\5.2\SysAction\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<!-- 显示联系人姓名的文本框 -->
<EditText
      android:id="@+id/show"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:editable="false"
      android:cursorVisible="false"
```

```
        />
    <!-- 显示联系人的电话的文本框 -->
    <EditText
        android:id="@+id/phone"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:editable="false"
        android:cursorVisible="false"
        />
    <Button
        android:id="@+id/bn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="查看联系人"
        />
    </LinearLayout>
```

上面的界面布局中包含了两个文本框，一个按钮，其中按钮用于浏览系统联系人列表并选择其中的联系人。两个文本框分别用于显示联系人的名字、电话号码。

注意：

由于该程序会用到系统的联系人应用，因此读者在运行该程序之前应该先进入 Android 系统自带的 Contacts 程序，并通过该程序添加几个联系人。



该程序的 Java 代码如下。

程序清单：codes\05\5.2\SysAction\src\org\crazyit\action\SysAction.java

```
public class SysAction extends Activity
{
    final int PICK_CONTACT = 0;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮绑定事件监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 创建 Intent
                Intent intent = new Intent();
                //设置 Intent 的 Action 属性
                intent.setAction(Intent.ACTION_GET_CONTENT);
                //设置 Intent 的 Type 属性
                intent.setType("vnd.android.cursor.item/phone");
                // 启动 Activity，并希望获取该 Activity 的结果
                startActivityForResult(intent, PICK_CONTACT);
            }
        });
    }
    @Override
    public void onActivityResult(int requestCode
        , int resultCode, Intent data)
```

```
    {
        super.onActivityResult(requestCode, resultCode, data);
        switch (requestCode)
        {
            case (PICK_CONTACT):
                if (resultCode == Activity.RESULT_OK)
                {
                    // 获取返回的数据
                    Uri contactData = data.getData();
                    CursorLoader cursorLoader = new CursorLoader(this
                        , contactData, null, null, null, null);
                    // 查询联系人信息
                    Cursor cursor = cursorLoader.loadInBackground();
                    // 如果查询到指定的联系人
                    if (cursor.moveToFirst())
                    {
                        String contactId = cursor.getString(cursor
                            .getColumnIndex(ContactsContract.
                            Contacts._ID));
                        // 获取联系人的名字
                        String name = cursor.getString(cursor
                            .getColumnIndexOrThrow(
                            ContactsContract.Contacts.DISPLAY_NAME));
                        String phoneNumber = "此联系人暂未输入电话号码";
                        //根据联系人查询该联系人的详细信息
                        Cursor phones = getContentResolver().query(
                            ContactsContract.CommonDataKinds.Phone.
                            CONTENT_URI,
                            null,
                            ContactsContract.CommonDataKinds.Phone.
                            CONTACT_ID
                                + " = " + contactId, null, null);
                        if (phones.moveToFirst())
                        {
                            //取出电话号码
                            phoneNumber = phones
                                .getString(phones
                                .getColumnIndex(ContactsContract
                                .CommonDataKinds.Phone.NUMBER));
                        }
                        // 关闭游标
                        phones.close();
                        EditText show = (EditText) findViewById(R.id.
                        show);
                        //显示联系人的名称
                        show.setText(name);
                        EditText phone = (EditText) findViewById(R.id.
                        phone);
                        //显示联系人的电话号码
                        phone.setText(phoneNumber);
                    }
                    // 关闭游标
                    cursor.close();
                }
                break;
        }
    }
}
```



提示：

该实例使用了后面章节中关于 ContentProvider 的知识，如果读者一时看不懂这个程序，可以参考后面章节的讲解进行理解。

运行上面的程序，单击程序界面中“查看联系人”按钮，程序将会显示如图 5.4 所示界面。

在图 5.4 所示的联系人列表中单击某个联系人，系统将会自动返回上一个 Activity，程序会在上一个 Activity 中显示所选联系人的名字和电话，如图 5.5 所示。

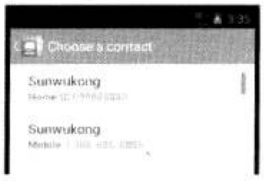


图 5.4　查看联系人



图 5.5　获取指定联系人的信息

上面的 Intent 对象除了设置 Action 属性之外，还设置了 Type 属性，关于 Intent 的 Type 属性的作用，等下将会进行更详细的介绍。

需要指出的是，由上面的程序需要查看系统联系人信息，因此不要忘了向该应用的 Android Manifest.xml 文件中增加相应的权限，也就是在 AndroidManifest.xml 文件中增加如下配置：

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

实例：返回系统 Home 桌面

接下来的示例将会提供一个按钮，当用户单击该按钮时，系统将会返回 Home 桌面，就像单击模拟器右边的按钮一样。这也需要通过 Intent 来实现，程序为 Intent 设置合适的 Action、合适的 Category 属性，并根据该 Intent 来启动 Activity 即可返回 Home 桌面。该示例程序如下。

程序清单：codes\05\5.2\ReturnHome\src\org\crazyit\intent\ReturnHome.java

```
public class ReturnHome extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                // 为 Intent 设置 Action、Category 属性
                intent.setAction(Intent.ACTION_MAIN);
                intent.addCategory(Intent.CATEGORY_HOME);
                startActivity(intent);
            }
```

```
            });
        }
    }
```

上面的程序中粗体字代码设置了 Intent 的 Action 为 android.intent.action.MAIN 字符串、Category 属性为 android.intent.category.HOME 字符串，满足该 Intent 的 Activity 其实就是 Android 系统的 Home 桌面。因此运行上面的程序时单击“返回桌面”按钮即可返回 Home 桌面。

5.2.4 Data、Type 属性与 intent-filter 配置

Data 属性通常用于向 Action 属性提供操作的数据。Data 属性接受一个 Uri 对象，一个 Uri 对象通常通过如下形式的字符串来表示：

```
content://com.android.contacts/contacts/1
tel:123
```

Uri 字符串总满足如下格式：

```
scheme://host:port/path
```

例如上面给出的 content://com.android.contacts/contacts/1，其中 content 是 scheme 部分，com.android.contacts 是 host 部分，port 部分被省略了，/contacts/1 是 path 部分。

Type 属性用于指定该 Data 所指定 Uri 对应的 MIME 类型，这种 MIME 类型可以是任何自定义的 MIME 类型，只要符合 abc/xyz 格式的字符串即可。

Data 属性与 Type 属性的关系比较微妙，这两个属性会相互覆盖，例如：

- 如果为 Intent 先设置 Data 属性，后设置 Type 属性，那么 Type 属性将会覆盖 Data 属性。
- 如果为 Intent 先设置 Type 属性，后设置 Data 属性，那么 Data 属性将会覆盖 Type 属性。
- 如果希望 Intent 既有 Data 属性，也有 Type 属性，应该调用 Intent 的 setDataAndType()方法。

下面的示例演示了 Intent 的 Data 与 Type 属性互相覆盖的的情形，该示例的界面布局文件很简单，只是定义了三个按钮，并为三个按钮绑定了事件处理函数。

下面是该示例的 Activity 代码。

程序清单：codes\05\5.2\DataTypeOverride\src\org\crazyit\intent\DataTypeOverride.java

```
public class DataTypeOverride extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void overrideType(View source)
    {
        Intent intent = new Intent();
        // 先为 Intent 设置 Type 属性
        intent.setType("abc/xyz");
        // 再为 Intent 设置 Data 属性，覆盖 Type 属性
        intent.setData(Uri.parse("lee://www.fkjava.org:8888/test"));
```

```
            Toast.makeText(this, intent.toString(), Toast.LENGTH_LONG).show();
        }
        public void overrideData(View source)
        {
            Intent intent = new Intent();
            // 先为 Intent 设置 Data 属性
            intent.setData(Uri.parse("lee://www.fkjava.org:8888/mypath"));
            // 再为 Intent 设置 Type 属性，覆盖 Data 属性
            intent.setType("abc/xyz");
            Toast.makeText(this, intent.toString(), Toast.LENGTH_LONG).show();
        }
        public void dataAndType(View source)
        {
            Intent intent = new Intent();
            // 同时设置 Intent 的 Data、Type 属性
            intent.setDataAndType(Uri.parse("lee://www.fkjava.org:8888/mypath"),
                "abc/xyz");
            Toast.makeText(this, intent.toString(), Toast.LENGTH_LONG).show();
        }
}
```

上面的三个事件监听方法分别为 Intent 设置了 Data、Type 属性，第一个事件监听方法先设置 Type 属性，再设置 Data 属性，这将导致 Data 属性覆盖 Type 属性，单击按钮激发该事件监听方法，将可以看到如图 5.6 所示的 Toast 输出。

从图 5.6 可以看出，此时的 Intent 只有 Data 属性，Type 属性被覆盖了。

上面的示例中第二个事件监听方法先设置了 Data 属性、再设置了 Type 属性，这将导致 Type 属性覆盖 Data 属性，单击按钮激发该事件监听方法，将可以看到如图 5.7 所示输出。



图 5.6　Data 属性覆盖 Type 属性

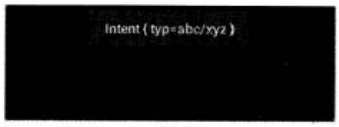


图 5.7　Type 属性覆盖 Data 属性

上面的示例中第三个事件监听方法同时设置了 Data、Type 属性、这样该 Intent 中才会同时具有 Data、Type 属性。

在 AndroidManifest.xml 文件中为组件声明 Data、Type 属性都通过<data.../>元素，<data.../>元素的格式如下：

```
<data android:mimeType=""
    android:scheme=""
    android:host=""
    android:port=""
    android:path=""
    android:pathPrefix=""
    android:pathPattern="" />
```

上面的<data.../>元素支持如下属性。

- mimeType：用于声明该组件所能匹配的 Intent 的 Type 属性。
- scheme：用于声明该组件所能匹配的 Intent 的 Data 属性的 scheme 部分。
- host：用于声明该组件所能匹配的 Intent 的 Data 属性的 host 部分。
- port：用于声明该组件所能匹配的 Intent 的 Data 属性的 port 部分。
- path：用于声明该组件所能匹配的 Intent 的 Data 属性的 path 部分。
- pathPrefix：用于声明该组件所能匹配的 Intent 的 Data 属性的 path 前缀。

- pathPattern：用于声明该组件所能匹配的 Intent 的的 Data 属性的 path 字符串模板。

Intent 的 Type 属性也用于指定该 Intent 的要求，必须对应组件中<intent-filter.../>元素中<data.../>子元素的 mimeType 属性与此相同，才能启动该组件。

Intent 的 Data 属性则略有差异，程序员为 Intent 指定 Data 属性时，Data 的属性的 Uri 对象实际上可分为 scheme、host、port 和 path 部分，此时并不要求被启动组件的<intent-filter.../>中<data.../>子元素的 android:scheme、android:host、android:port、android:path 完全满足。

Data 属性的“匹配”过程则有些差别，它会先检查<intent-filter.../>里的<data.../>子元素，然后：

- 如果目标组件的<data.../>子元素只指定了 android:scheme 属性，那么只要 Intent 的 Data 属性的 scheme 部分与 android:scheme 属性值相同，即可启动该组件。
- 如果目标组件的<data.../>子元素只指定了 android:scheme、android:host 属性，那么只要 Intent 的 Data 属性的 scheme、host 部分与 android:scheme、android:host 属性值相同，即可启动该组件。
- 如果目标组件的<data.../>子元素指定了 android:scheme、android:host、android:port 属性，那么要求 Intent 的 Data 属性的 scheme、host、port 部分与 android:scheme、android:host、android:host 属性值相同，即可启动该组件。



提示：

如果<data.../>子元素只有 android:port 属性，没有指定 android:host 属性，那么 android:port 属性将不会起作用。

- 如果目标组件的<data.../>子元素只指定了 android:scheme、android:host、android:path 属性，那么只要求 Intent 的 Data 属性的 scheme、host、path 部分与 android:scheme、android:host、android:path 属性值相同，即可启动该组件。



提示：

如果<data.../>子元素只有 android:path 属性，没有指定 android:host 属性，那么 android:path 属性将不会起作用。

- 如果目标组件的<data.../>子元素指定了 android:scheme、android:host、android:port、android:path 属性，那么就要求 Intent 的 Data 属性的 scheme、host、port、path 部分依次与 android:scheme、android:host、android:port、android:path 属性值相同，才可启动该组件。

下面的示例测试了 Intent 的 Data 属性与<data.../>元素配置的关系，该示例依次配置了如下 5 个 Activity。

程序清单：codes\05\5.2\DataTypeAttr\AndroidManifest.xml

```
<activity
    android:icon="@drawable/ic_scheme"
    android:name=".SchemeActivity"
    android:label="指定 scheme 的 Activity">
    <intent-filter>
        <action android:name="xx" />
        <category android:name="android.intent.category.DEFAULT" />
        <!-- 只要 Intent 的 Data 属性的 scheme 是 lee，即可启动该 Activity -->
        <data android:scheme="lee" />
    </intent-filter>
```

```
</activity>
<activity
      android:icon="@drawable/ic_host"
      android:name=".SchemeHostPortActivity"
      android:label="指定 scheme、host、port 的 Activity">
      <intent-filter>
            <action android:name="xx" />
            <category android:name="android.intent.category.DEFAULT" />
            <!-- 只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org
            port 是 8888 即可启动该 Activity -->
            <data android:scheme="lee"
                  android:host="www.fkjava.org"
                  android:port="8888" />
      </intent-filter>
</activity>
<activity
      android:icon="@drawable/ic_sp"
      android:name=".SchemeHostPathActivity"
      android:label="指定 scheme、host、path 的 Activity">
      <intent-filter>
            <action android:name="xx" />
            <category android:name="android.intent.category.DEFAULT" />
            <!-- 只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org
            path 是/mypath，即可启动该 Activity -->
            <data android:scheme="lee"
                  android:host="www.fkjava.org"
                  android:path="/mypath" />
      </intent-filter>
</activity>
<activity
      android:icon="@drawable/ic_path"
      android:name=".SchemeHostPortPathActivity"
      android:label="指定 scheme、host、port、path 的 Activity">
      <intent-filter>
            <action android:name="xx" />
            <category android:name="android.intent.category.DEFAULT" />
            <!-- 需要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org
            port 是 8888，且 path 是/mypath，才可启动该 Activity -->
            <data android:scheme="lee"
                  android:host="www.fkjava.org"
                  android:port="8888"
                  android:path="/mypath"/>
      </intent-filter>
</activity>
<activity
      android:icon="@drawable/ic_type"
      android:name=".SchemeHostPortPathTypeActivity"
      android:label="指定 scheme、host、port、path、type 的 Activity">
      <intent-filter>
            <action android:name="xx"/>
            <category android:name="android.intent.category.DEFAULT" />
            <!-- 需要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org
            port 是 8888，且 path 是/mypath
            且 type 是 abc/xyz，才可启动该 Activity -->
            <data android:scheme="lee"
                  android:host="www.fkjava.org"
                  android:port="8888"
                  android:path="/mypath"
                  android:mimeType="abc/xyz"/>
      </intent-filter>
</activity>
```

上面的配置文件中配置了 5 个 Activity，这 5 个 Activity 的实现类都非常简单，它们都仅在界面上显示一个 TextView，并不显示其他内容。关于这 5 个 Activity 的<data.../>子元素配置的说明如下。

- 第 1 个 Activity：只要 Intent 的 Data 属性的 scheme 是 lee，即可启动该 Activity。
- 第 2 个 Activity：只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org、port 是 8888，即可启动该 Activity。
- 第 3 个 Activity：只要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org、path 是/mypath，即可启动该 Activity。
- 第 4 个 Activity：需要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org、port 是 8888，且 path 是/mypath，才可启动该 Activity。
- 第 5 个 Activity：需要 Intent 的 Data 属性的 scheme 是 lee，且 host 是 www.fkjava.org、port 是 8888，且 path 是/mypath，且 type 是 abc/xyz，才可启动该 Activity。

注意：

上面配置 Activity 的<intent-filter.../>元素时，<action.../>子元素的 name 属性是随意指定的，这是必需的。如果希望<data.../>子元素能正常起作用，至少要配置一个<action.../>子元素，但该子元素的 android:name 属性值可以是任意的字符串。

下面是第一个启动 Activity 的方法：

```
public void scheme(View source)
{
    Intent intent = new Intent();
    // 只设置 Intent 的 Data 属性
    intent.setData(Uri.parse("lee://www.crazyit.org:1234/test"));
    startActivity(intent);
}
```

由于上面 Intent 的 Data 属性，只有 scheme 为 lee，也就是只有第 1 个 Activity 符合条件，因此通过该方法启动 Activity 时，将可看到启动如图 5.8 所示的 Activity。



图 5.8 只有 scheme 匹配的 Activity

下面是第 2 个启动 Activity 的方法：

```
public void schemeHostPort(View source)
{
    Intent intent = new Intent();
    // 只设置 Intent 的 Data 属性
    intent.setData(Uri.parse("lee://www.fkjava.org:8888/test"));
    startActivity(intent);
}
```

由于上面 Intent 的 Data 属性，只有 scheme 为 lee，也就是有第一个 Activity 符合条件；且该 Intent 的 Data 属性的 host 为 www.fkjava.org、port 为 8888，因此第二个 Activity 也符合条件。通过该方法启动 Activity 时将可看到启动如图 5.9 所示的选择 Activity 的界面。

下面是第 3 个启动 Activity 的方法：

```
public void schemeHostPath(View source)
```

```
{
    Intent intent = new Intent();
    // 只设置 Intent 的 Data 属性
    intent.setData(Uri.parse("lee://www.fkjava.org:1234/mypath"));
    startActivity(intent);
}
```

由于上面 Intent 的 Data 属性，只有 scheme 为 lee，也就是有第一个 Activity 符合条件；且该 Intent 的 Data 属性的 host 为 www.fkjava.org、path 为/mypath，因此第三个 Activity 也符合条件。通过该方法启动 Activity 时，将可看到启动如图 5.10 所示的选择 Activity 的界面。



图 5.9　scheme、host、port 匹配的 Activity



图 5.10　scheme、host、path 匹配的 Activity

下面是第 4 个启动 Activity 的方法。

```
public void schemeHostPortPath(View source)
{
    Intent intent = new Intent();
    // 只设置 Intent 的 Data 属性
    intent.setData(Uri.parse("lee://www.fkjava.org:8888/mypath"));
    startActivity(intent);
}
```



图 5.11　scheme、host、port、path 匹配的 Activity

由于上面 Intent 的 Data 属性，只有 scheme 为 lee，也就是有第一个 Activity 符合条件；且该 Intent 的 Data 属性的 host 为 www.fkjava.org、port 为 8888，因此第二个 Activity 也符合条件；且该 Intent 的 Data 属性的 host 为 www.fkjava.org、path 为/mypath，因此第三个 Activity 也符合条件；且该 Intent 的 Data 属性的 host 为 www.fkjava.org、port 为 8888、path 为/mypath，因此第四个 Activity 也符合条件。通过该方法启动 Activity 时，将可看到启动如图 5.11 所示的选择 Activity 的界面。

下面是第 5 个启动 Activity 的方法：

```
public void schemeHostPortPathType(View source)
{
    Intent intent = new Intent();
    // 同时设置 Intent 的 Data、Type 属性
    intent.setDataAndType(Uri.parse("lee://www.fkjava.org:8888/mypath")
        , "abc/xyz");
    startActivity(intent);
}
```

上面的 Intent 不仅指定了 Data 属性，也指定了 Type 属性，此时符合条件的只有第五个 Activity，通过该方法启动 Activity 时，将可看到启动如图 5.12 所示的 Activity。

实例：使用 Action、Data 属性启动系统 Activity

一旦为 Intent 同时指定了 Action、Data 属性，那么 Android 将可根据指定的数据类型来启动特定的应用程序，并对指定数据执行相应的操作。



图 5.12 scheme、host、port、path、type 匹配的 Activity

下面是几个 Action 属性、Data 属性的组合。

- ACTION_VIEW content://com.android.contacts/contacts/1：显示标识为 1 的联系人的信息。
- ACTION_EDIT content://com.android.contacts/contacts/1：编辑标识为 1 的联系人的信息。
- ACTION_DIAL content://com.android.contacts/contacts/1：显示向标识为 1 的联系人拨号的界面。
- ACTION_VIEW tel:123：显示向指定号码 123 拨号的界面。
- ACTION_DIAL tel:123：显示向指定号码 123 拨号的界面。
- ACTION_VIEW content://contacts/people/：显示所有联系人列表的信息，通过这种组合可以非常方便地查看系统联系人。

下面的程序示范通过同时指为 Intent 指定 Action、Data 属性来启动特定程序并操作相应的数据。下面的程序的界面很简单，它只包含两个按钮，其中一个按钮用于浏览指定网页，一个按钮用于编辑指定联系人信息。

程序清单：codes\05\5.2\ActionData\src\org\crazyit\intent\ActionData.java

```
public class ActionData extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button bn = (Button) findViewById(R.id.bn);
        // 为 bn 按钮添加一个监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent
                Intent intent = new Intent();
                String data = "http://www.crazyit.org";
                // 根据指定字符串解析出 Uri 对象
                Uri uri = Uri.parse(data);
                // 为 Intent 设置 Action 属性
                intent.setAction(Intent.ACTION_VIEW);
                // 设置 Data 属性
                intent.setData(uri);
                startActivity(intent);
            }
        });
        Button edit = (Button) findViewById(R.id.edit);
        // 为 edit 按钮添加一个监听器
        edit.setOnClickListener(new OnClickListener()
```

```
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent
                Intent intent = new Intent();
                // 为 Intent 设置 Action 属性（动作为：编辑）
                intent.setAction(Intent.ACTION_EDIT);
                String data = "content://com.android.contacts/contacts/1";
                // 根据指定字符串解析出 Uri 对象
                Uri uri = Uri.parse(data);
                // 设置 Data 属性
                intent.setData(uri);
                startActivity(intent);
            }
        });
        Button call = (Button) findViewById(R.id.call);
        // 为 edit 按钮添加一个监听器
        call.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent
                Intent intent = new Intent();
                // 为 Intent 设置 Action 属性（动作为：拨号）
                intent.setAction(Intent.ACTION_DIAL);
                String data = "tel:13800138000";
                // 根据指定字符串解析出 Uri 对象
                Uri uri = Uri.parse(data);
                // 设置 Data 属性
                intent.setData(uri);
                startActivity(intent);
            }
        });
    }
}
```

运行上面的程序，单击第一个按钮，该按钮单击时启动 Intent（Action= Intent.ACTION_VIEW，Data= http://www. crazyit.org）对应的 Activity，将看到打开 www.crazyit.org 的界面，如图 5.13 所示。

单击第二个按钮，该按钮单击时启动 Intent（Action= Intent.ACTION_EDIT，Data= content://com.android.contacts/ contacts/1）对应的 Activity，将看到编辑标识为 1 的联系人的界面，如图 5.14 所示。



图 5.13　使用 Action、Data 打开指定网页



图 5.14　使用 Action、Data 属性编辑指定联系人

单击第三个按钮，该按钮单击时启动 Intent（Action= Intent.ACTION_DIAL，Data=tel:

13800138000）对应的 Activity，将看到程序向 13800138000 拨号的界面，如图 5.15 所示。

5.2.5 Extra 属性



图 5.15 使用 Action、Data 属性向指定号码拨号

Intent 的 Extra 属性通常用于在多个 Action 之间进行数据交换，Intent 的 Extra 属性值应该是一个 Bundle 对象，Bundle 对象的就像一个 Map 对象，它可以存入多组 key-value 对，这样可以就通过 Intent 在不同 Activity 之间进行数据交换了。关于 Extra 属性的用法前面已有示例，故此处不再赘述。

5.2.6 Flag 属性

Intent 的 Flag 属性用于为该 Intent 添加一些额外的控制旗标，Intent 可调用 addFlags()方法来为 Intent 添加控制旗标。



提示： 前面介绍启用 ActionBar 的程序图标返回主 Activity 时已经用到了 Flag 属性。比如前面介绍的 Intent.FLAG_ACTIVITY_CLEAR_TOP 旗标可用于清除当前 Activity 栈中的 Activity。

除此之外，Intent 还包含了如下常用的 Flag 旗标。

- FLAG_ACTIVITY_BROUGHT_TO_FRONT：如果通过该 Flag 启动的 Activity 已经存在，下次再次启动时，将只是将该 Activity 带到前台。例如现在 Activity 栈中有 Activity A，此时以该旗标启动 Activity B（即 Activity B 是以 FLAG_ACTIVITY_BROUGHT_TO_FRONT 旗标启动的），然后在 Activity B 中启动 C、D，如果此时在 Activity D 中再启动 B，将直接把 Activity 栈中的 Activity B 带到前台。此时 Activity 栈中情形是 A、C、D、B。
- FLAG_ACTIVITY_CLEAR_TOP：该 Flag 相当于加载模式中的 singleTask，通过这种 Flag 启动的 Activity 将会把要启动的 Activity 之上的 Activity 全部弹出 Activity 栈。例如，Activity 栈中包含 A、B、C、D 这 4 个 Activity，如果采用该 Flag 从 Activity D 跳转到 Activity B，此时 Activity 栈中只包含 A、B 两个 Activity。
- FLAG_ACTIVITY_NEW_TASK：默认的启动旗标，该旗标控制重新创建一个新的 Activity。
- FLAG_ACTIVITY_NO_ANIMATION：该旗标会控制启动 Activity 时不使用过渡动画。
- FLAG_ACTIVITY_NO_HISTORY：该旗标控制被启动的 Activity 将不会保留在 Activity 栈中。例如 Activity 栈中原来有 A、B、C 这三个 Activity，此时在 Activity C 中以该 Flag 启动 Activity D，Activity D 再启动 Activity E，此时 Activity 中只有 A、B、C、E 这 4 个 Activity，Activity D 不会保留在 Actvity 栈中。
- FLAG_ACTIVITY_REORDER_TO_FRONT：该 Flag 控制如果当前已有该 Activity，直接将该 Activity 带到前台。例如现在 Activity 栈中有 A、B、C、D 这 4 个 Activity，如果使用 FLAG_ACTIVITY_REORDER_TO_FRONT 旗标来启动 Activity B，那么启动后的 Activity 栈中情形为 A、C、D、B。

- FLAG_ACTIVITY_SINGLE_TOP：该 Flag 相当于加载模式中的 singleTop 模式，例如原来 Activity 栈中有是 A、B、C、D 这 4 个 Activity，在 Activity D 中再次启动 Activity D，Activity 栈中依然还是 A、B、C、D 这 4 个 Activity。

Android 提供为 Intent 提供了大量的 Flag，每个 Flag 都有其特定的功能，具体请参考关于 Intent 的 API 文档。

5.3 使用 Intent 创建 Tab 页面

前面已经介绍了如何使用 TabActivity 来创建 Activity 布局，前面添加 Tab 页面使用了 TabHost.TabSpec 如下方法。

- setContent(int viewId)：直接将指定 View 组件设置成 Tab 页的 Content。

实际上 TabHost.TabSpec 还提供了一个如下方法。

- setContent(Intent intent)：直接将指定 Intent 对应的 Activity 设置成 Tab 页的 Content。

例如如下 Activity 代码。

程序清单：codes\05\5.3\IntentTab\src\org\crazyit\intent\IntentTab.java

```
public class IntentTab extends TabActivity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取该 Activity 里面的 TabHost 组件
        TabHost tabHost = getTabHost();
        // 使用 Intent 添加第一个 Tab 页面
        tabHost.addTab(tabHost
            .newTabSpec("tab1")
            .setIndicator("已接电话",
                getResources().getDrawable(R.drawable.ic_launcher))
            .setContent(new Intent(this, BeCalledActivity.class)));
        // 使用 Intent 添加第二个 Tab 页面
        tabHost.addTab(tabHost.newTabSpec("tab1")
            .setIndicator("呼出电话")
            .setContent(new Intent(this, CalledActivity.class)));
        // 使用 Intent 添加第三个 Tab 页面
        tabHost.addTab(tabHost.newTabSpec("tab1")
            .setIndicator("未接电话")
            .setContent(new Intent(this, NoCallActivity.class)));
    }
}
```

上面的程序中三行粗体字代码用于为 TabHost.TabSpec 设置 Content，三行粗体字代码分别传入了三个 Intent 对象，这意味着该 TabHost 将直接以三个 Activity 类作为 Tab 页。上面的程序还需要 BeCalledActivity、CalledActivity、NoCallActivity 三个 Activity 类，不过这三个 Activity 都很简单，故此处不再给出。运行上面的程序将看到如图 5.16 所示的界面。



图 5.16　使用 Intent 创建 Tab 页

5.4 本章小结

本章主要介绍了 Android 系统中 Intent 的功能和用法，当 Android 应用需要启动某个组件时，总需要借助于 Intent 来实现。不管是启动 Activity，还是启动 Service、BroadcastReceiver 组件，Android 系统都是由 Intent 来实现的。简单地说，Android 使用 Intent 封装了应用程序的“启动意图”，但这种“意图”并未直接与任何程序组件耦合，通过这种方式即可很好地提供系统的可扩展性和可维护性。学习本章需要重点掌握 Intent 的 Component、Action、Category、Data、Type 各属性的功能和用法，并掌握如何在 AndroidManifest.xml 文件中配置 <intent-filter.../>元素。

CHAPTER

6

第6章 Android应用的资源

本章要点

- Android 应用的资源和作用
- Android 应用的资源的存储方式
- 在 XML 布局文件中使用资源
- 在 Java 程序中使用资源
- 使用字符串资源
- 使用颜色资源
- 使用尺寸资源
- 使用数组资源
- 使用图片资源
- 使用各种 Drawable 资源
- 使用原始 XML 资源
- 使用布局资源
- 使用菜单资源
- 使用样式和主题资源
- 使用属性资源
- 使用原始资源
- 使用资源进行程序国际化

经过前面的介绍，相信读者对 Android 应用已有了大致的了解。如果从物理存在形式来分，Android 应用的源代码大致可分为如下三大类。

- 界面布局文件：XML 文件，文件中每个标签都对应于相应的 View 标签。
- Java 源文件：应用中的 Activity、Service、BroadcastReceiver、ContentProvider 四大组件都是采用 Java 代码来实现的。
- 资源文件：主要以各种 XML 为主，还可包括*.png、*.jpg、*.gif 图片资源。

在传统Java应用中，初学者很容易犯一个错误：直接在Java源代码中使用如“crazyit.org”、"hello"这样的字符串，或直接使用 123、0.9 这样的数值，而且不添加任何注释。过了一段时间后，即使他自己再去看原来写的程序代码，一时之间，也无法理解其中"crazyit.org"、"hello"字符串，123、0.9 等数值的含义，这种方式就大大增加了程序的维护成本。这种直接在代码中定义的 123、0.9 等数值，也被称为“魔术数值”（就像表演魔术一样，其他人都搞不懂）。

为了改善这种情况，有经验的开发者会专门定义一个或多个接口或类，然后在其中以常量的形式来定义程序中用的所有字符串、数值等，这些常量的名称十分明确，如 ResultSet.TYPE_FORWARD_ONLY，相信有经验的读者一看到这个常量就大致明白了它的含义。这样的方式就可以很好地提高程序的可维护性。

使用接口或类的形式来定义程序中用的字符串、数值虽然已经部分提高了程序的解耦，但后期维护、进一步开发时，开发人员还得去 Java“代码海”中打捞那些定义字符串常量、数值常量的位置，因此还有可以提高的地方。

Android 应用则对这种字符串常量、数值常量的定义做了进一步改进：Android 允许把应用中用到的各种资源：字符串资源、颜色资源、数组资源、菜单资源等都集中放到 res 目录中定义，应用程序则直接使用这些资源中定义的值。

Android 应用下除了 res 目录用于存放资源之外，assets 目录也用于存放资源。一般来说，assets 目录下存放的资源代表应用无法直接访问的原生资源，应用程序需要通过 AssetManager 以二进制流的形式来读取资源。而 res 目录下的资源，Android SDK 会在编译该应用时，自动在 R.java 文件中为这些资源创建索引，程序可直接通过 R 资源清单类进行访问。

我们前面介绍的很多示例都是直接将字符串值直接写在界面布局文件中或 Java 程序代码中，实际上那并不是一种好的方式。只是前面还未详细介绍 Android 应用的资源，为了避免读者产生畏难心理，并未使用资源文件而已。

6.1 资源的类型及存储方式

Android 应用资源可分为两大类：

- 无法通过 R 清单类访问的原生资源，保存在 assets 目录下。
- 可通过 R 资源清单类访问的资源，保存在 res 目录下。

大部分时候提到 Android 应用资源时，往往都是指位于 res 目录下的应用资源，Android SDK 会在编译该应用时在 R 类中为它们创建对应的索引项。

6.1.1 资源的类型以及存储方式

Android 要求在 res 目录下用不同的子目录来保存不同的应用资源，表 6.1 大致显示了

Android 不同资源在/res 目录下的存储方式。

表 6.1　Android 应用资源的存储

目　录	存放的资源
/res/animator/	存放定义属性动画的 XML 文件
/res/anim/	存放定义补间动画的 XML 文件
/res/color/	存放定义不同状态下颜色列表的 XML 文件
/res/drawable/	该目录下存放各种位图文件（如*.png、*.9.png、*.jpg、*.gif）等。除此之外也可是能编译成如下各种 Drawable 对象的 XML 文件： ➢ BitmapDrawable ➢ NinePatchDrawable 对象 ➢ StateListDrawable 对象 ➢ ShapeDrawable 对象 ➢ AnimationDrawable 对象 ➢ Drawable 的其他各种子类的对象
/res/layout/	存放各种用户界面的布局文件
/res/menu/	存放为应用程序定义各种菜单的资源，包括选项菜单、子菜单、上下文菜单资源
/res/raw/	该目录下存放任意类型的原生资源（比如音频文件、视频文件等）。在 Java 代码中可通过调用 Resources 对象的 openRawResource(int id)方法来获取该资源的二进制输入流。实际上，如果应用程序需要使用原生资源，推荐把这些原生资源保存到/assets 目录下，然后在应用程序中使用 AssetManager 来访问这些资源
/res/values/	存放各种简单值的 XML 文件。这些简单值包括字符串值、整数值、颜色值、数组等。 字符串值、整数值、颜色值、数组等各种值都是存放在该目录下的，而且这些资源文件的根元素都是<resources.../>元素，当我们为该<resources.../>元素添加不同的子元素则代表不同的资源，例如， ➢ string/integer/bool 子元素：代表添加一个字符串值、整数值或 boolean 值。 ➢ color 子元素：代表添加一个颜色值。 ➢ array 子元素或 string-array、int-array 子元素：代表添加一个数组。 ➢ style 子元素：代表添加一个样式。 ➢ dimen：代表添加一个尺寸。 ➢ … 由于各种简单值都可定义在/res/values/目录下的资源文件中，如果在同一份资源文件中定义各种值，势必增加程序维护的难度。为此，Android 建议使用不同的文件来存放不同类型的值，例如， ➢ arrays.xml：定义数组资源。 ➢ colors.xml：定义颜色值资源。 ➢ dimens.xml：定义尺寸值资源。 ➢ strings.xml：定义字符串资源。 ➢ styles.xml：定义样式资源
/res/xml/	任意的原生 XML 文件。这些 XML 文件可在 Java 代码中使用 Resources.getXML()方法进行访问

一旦将应用程序的各种资源分别保存在 Android 应用的/res 目录下，接下来既可以在 Java 程序中使用这些资源，也可以在其他 XML 资源中使用这些资源。



提示： 对于 Android 4.2 版本而言，应用的/res/目录并没有包含 drawable 子目录，而是提供了 drawable-ldpi（低分辨率）、drawable-mdpi（中等分辨率）、drawable-hdpi（高分辨率）、drawable-xhdpi（超高分辨率）4 个子目录，其实这 4 个子目录的作用就相当于 drawable 子目录。drawable-ldpi、drawable-mdpi、drawable-hdpi 三个子目录下存放的图片文件的文件名完全相同，只是分辨率不同——这种做法可以让系统根据屏幕分辨率来选择相应的图片。大部分程序、系统将会选择 drawable-mdpi 目录下的图片文件，如果系统使用高分屏，那么系统将会选择 drawable-hdpi 目录下的图片文件。

6.1.2 使用资源

在 Android 应用中使用资源可分为在 Java 代码和 XML 文件中使用资源，其中 Java 代码用于为 Android 应用定义四大组件，而 XML 文件中则用于为 Android 应用定义各种资源。

1. 在 Java 代码中使用资源清单项

由于 Android SDK 会在编译应用时在 R 类中为/res 目录下所有资源创建索引项，因此在 Java 代码中访问资源主要通过 R 类来完成。其完整的语法格式为：

```
[<package_name>.]R.<resource_type>.<resource_name>
```

上面语法格式中各成分的说明如下。

- **<package_name>**：指定 R 类所在包，实际上就是使用全限定类名。当然，如果在 Java 程序中导入 R 类所在包，就可以省略包名。
- **<resource_type>**：R 类中代表不同资源类型的子类，例如 string 代表字符串资源。
- **<resource_name>**：指定资源的名称。该资源名称可能是无后缀的文件名（如图片资源），也可能是 XML 资源元素中由 android:name 属性所指定的名称。

例如如下代码片段：

```
// 从 drawable 资源中加载图片，并设为该窗口的背景
getWindow().setBackgroundDrawableResource(R.drawable.back) ;
// 从 string 资源中获取指定字符串资源，并设置该窗口的标题
getWindow().setTitle(getResources().getText(R.string.main_title));
// 获取指定的 TextView 组件，并设置该组件显示 string 资源中的指定字符串资源
TextView msg = (TextView) findViewById(R.id.msg);
msg.setText(R.string.hello_message);
```

2. 在 Java 代码中访问实际资源

R 资源清单类为所有的资源都定义了一个资源清单项，但这个清单项只是一个 int 类型的值，并不是实际的资源对象。大部分情况下，Android 应用的 API 允许直接使用 int 类型的资源清单项代替应用资源。

但有些时候，程序也需要使用实际的 Android 资源，为了通过资源清单项目来获取实际资源，可以借助于 Android 提供的 Resources 类。



提示： 笔者把 Resources 类称为“Android 资源访问总管家”，Resources 类提供了大量的方法来根据资源清单 ID 获取实际资源。

Resources 主要提供了如下两类方法。

- getXxx(int id)：根据资源清单 ID 来获取实际资源。
- getAssets()：获取访问/assets/目录下资源的 AssetManager 对象。

Resources 由 Context 调用 getResources()方法来获取。

下面的代码片段示范了如何通过 Resources 获取实际字符串资源。

```
// 直接调用 Activity 的 getResources()方法来获取 Resources 对象
Resources res = getResources();
// 获取字符串资源
String mainTitle = res.getText(R.string.main_title)
// 获取 Drawable 资源
Drawable logo = res.getDrawable(R.drawable.logo);
// 获取数组资源
int[] arr = res.getIntArray(R.array.books);
```

3. 在 XML 代码中使用资源

当定义 XML 资源文件时，其中的 XML 元素可能需要指定不同的值，这些值就可设置为已定义的资源项。在 XML 代码中使用资源的完整语法格式为：

```
@[<package_name>:]<resource_type>/<resource_name>
```

上面语法格式中各成分的说明如下。

- <package_name>：指定资源类所在应用的包。如果所引用的资源和当前资源位于同一个包下，则<package_name>可以省略。
- <resource_type>：R 类中代表不同资源类型的子类。
- <resource_name>：指定资源的名称。该资源名称可能是无后缀的文件名（如图片资源），也可能是 XML 资源元素中由 android:name 属性所指定的名称。

例如如下代码片段在一份文件中定义了两种资源：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="red">#ff00</color>
    <string name="hello">Hello!</string>
</resources>
```

接下来与它位于同一包中的 XML 资源文件就可通过如下方式来使用资源：

```
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/red"
    android:text="@string/hello" />
```

接下来将会对各种资源分别进行详细阐述。

6.2 使用字符串、颜色、尺寸资源

字符串资源、颜色资源、尺寸资源，它们对应的 XML 文件都将位于/res/values 目录下，它们默认的文件名，以及在 R 类中对应的内部类如表 6.2 所示。

表 6.2 字符串、颜色、尺寸资源表

资源类型	资源文件的默认名	对应于 R 类中的内部类的名称
字符串资源	/res/values/strings.xml	R.string
颜色资源	/res/values/colors.xml	R.color
尺寸资源	/res/values/dimens.xml	R.dimen

下面会通过示例来介绍字符串资源、颜色资源和尺寸资源的用法。

6.2.1 颜色值的定义

Android 中的颜色值是通过红（Red）、绿（Green）、蓝（Blue）三原色，以及一个透明度（Alpha）值来表示的，颜色值总是以井号（#）开头，接下来就是 Alpha-Red-Green-Blue 的形式。其中 Alpha 值可以省略，如果省略了 Alpha 值，那么该颜色默认是完全不透明的。

Android 颜色值支持常见的 4 种形式。

- #RGB：分别指定红、绿、蓝三原色的值（只支持 0～f 这 16 级颜色）来代表颜色。
- #ARGB：分别指定红、绿、蓝三原色的值（只支持 0～f 这 16 级颜色）及透明度（只支持 0～f 这 16 级透明度）来代表颜色。
- #RRGGBB：分别指定红、绿、蓝三原色的值（支持 00～ff 这 256 级颜色）来代表颜色。
- #AARRGGBB：：分别指定红、绿、蓝三原色的值（支持 00～ff 这 256 级颜色）以及透明度（支持 00～ff 这 256 级透明度）来代表颜色。

上面 4 种形式中，A、R、G、B 都代表一个十六进制的数，其中 A 代表透明度，R 代表红色的数值、G 代表绿色数值、B 代表蓝色数值。

注意：

关于颜色与三原色的知识，请读者自行参考光学方面的知识。此处笔者粗略介绍一下三原色理论：白色光大致可“分解”为红、绿、蓝三种光，红、绿、蓝三种光可合并成白光。当红、绿、蓝都是最大值时，三种光合并就会变成白光；当三种光的值相等，但不是最大值时，三种光将会合并成灰色光；如果其中一种光或两种光的值更亮，那么三种光合并就会产生彩色的光。



6.2.2 定义字符串、颜色、尺寸资源文件

字符串资源文件位于/res/values 目录下，字符串资源文件的根元素是<resources...>，该元素里每个<string.../>子元素定义一个字符串常量，其中<string.../>元素的 name 属性指定该常量的名称，<string.../>元素开始标签和结束标签之间的内容代表字符串值，如以下代码所示：

```
<!-- 定义一个字符串，名称为 hello，字符串值为 Hello World, ValuesResTest! -->
<string name="hello">Hello World, ValuesResTest!</string>
```

如下文件是该示例的字符串资源文件。

程序清单：codes\06\6.2\ValuesResTest\res\values\strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="hello">Hello World, ValuesResTest!</string>
    <string name="app_name">字符串、数字、尺寸资源</string>
    <string name="c1">F00</string>
    <string name="c2">0F0</string>
    <string name="c3">00F</string>
    <string name="c4">0FF</string>
    <string name="c5">F0F</string>
    <string name="c6">FF0</string>
    <string name="c7">07F</string>
    <string name="c8">70F</string>
    <string name="c9">F70</string>
</resources>
```

上面的程序代码中每个<string.../>元素定义一个字符串，其中<string.../>元素的 name 属性定义字符串的名称，<string>与</string>中间的内容就是该字符串的值。

颜色资源文件位于/res/values 目录下，颜色资源文件的根元素是<resources...>，该元素里每个<color.../>子元素定义一个字符串常量，其中<color.../>元素的 name 属性指定该颜色的名称，<color.../>元素开始标签和结束标签之间的内容代表颜色值，如以下代码所示：

```
<!-- 定义一个颜色，名称为 c1，颜色为红色-->
<color name="c1">#F00</color>
```

如下文件是该示例的颜色资源文件。

程序清单：codes\06\6.2\ValuesResTest\res\values\colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="c1">#F00</color>
    <color name="c2">#0F0</color>
    <color name="c3">#00F</color>
    <color name="c4">#0FF</color>
    <color name="c5">#F0F</color>
    <color name="c6">#FF0</color>
    <color name="c7">#07F</color>
    <color name="c8">#70F</color>
    <color name="c9">#F70</color>
</resources>
```

上面的程序代码中每个<color.../>元素定义一个字符串，其中<color.../>元素的 name 属性定义颜色的名称，<color>与</color>中间的内容就是该颜色的值。

尺寸资源文件位于/res/values 目录下，尺寸资源文件的根元素是<resources...>，该元素里每个<dimen.../>子元素定义一个尺寸常量，其中<dimen.../>元素的 name 属性指定该尺寸的名称，<dimen.../>元素开始标签和结束标签之间的内容代表尺寸值，如以下代码所示。

程序清单：codes\06\6.2\ValuesResTest\res\values\dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="spacing">8dp</dimen>
    <!-- 定义 GridView 组件中每个单元格的宽度、高度 -->
    <dimen name="cell_width">60dp</dimen>
    <dimen name="cell_height">66dp</dimen>
    <!-- 定义主程序的标题的字体大小 -->
    <dimen name="title_font_size">18sp</dimen>
</resources>
```

上面三份资源文件分别定义了字符串、颜色、尺寸资源，应用程序接下来既可在 XML 文件中使用这些资源，也可在 Java 代码中使用这些资源。

6.2.3 使用字符串、颜色、尺寸资源

正如前面介绍的，在 XML 文件中使用资源按如下语法格式：

```
@[<package_name>:]<resource_type>/<resource_name>
```

下面程序的界面布局中大量使用了前面定义的资源。

程序清单：codes\06\6.2\ValuesResTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<!-- 使用字符串资源，尺寸资源 -->
<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/app_name"
      android:gravity="center"
      android:textSize="@dimen/title_font_size"
/>
<!-- 定义一个 GridView 组件，使用尺寸资源中定义的长度来指定水平间距、垂直间距 -->
<GridView
      android:id="@+id/grid01"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:horizontalSpacing="@dimen/spacing"
      android:verticalSpacing="@dimen/spacing"
      android:numColumns="3"
      android:gravity="center">
</GridView>
</LinearLayout>
```

上面的程序中粗体字代码就是使用字符串资源、尺寸资源的代码。

在 Java 代码中使用资源按如下语法格式：

```
[<package_name>.R.<resource_type>.<resource_name>
```

下面的 Java 程序同时使用了上面定义的三种资源。

程序清单：codes\06\6.2\ValuesResTest\src\org\crazyit\res\ValuesResTest.java

```
public class ValuesResTest extends Activity
{
      // 使用字符串资源
      int[] textIds = new int[]
      {
            R.string.c1 , R.string.c2 , R.string.c3 ,
            R.string.c4 , R.string.c5 , R.string.c6 ,
            R.string.c7 , R.string.c8 , R.string.c9
      };
      // 使用颜色资源
      int[] colorIds = new int[]
      {
            R.color.c1 , R.color.c2 , R.color.c3 ,
            R.color.c4 , R.color.c5 , R.color.c6 ,
            R.color.c7 , R.color.c8 , R.color.c9
      };
```

```
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 创建一个 BaseAdapter 对象
    BaseAdapter ba = new BaseAdapter()
    {
        @Override
        public int getCount()
        {
            // 指定一共包含 9 个选项
            return textIds.length;
        }

        @Override
        public Object getItem(int position)
        {
            // 返回指定位置的文本
            return getResources().getText(textIds[position]);
        }

        @Override
        public long getItemId(int position)
        {
            return position;
        }
        // 重写该方法，该方法返回的 View 将作为 GridView 的每个格子
        @Override
        public View getView(int position
            , View convertView, ViewGroup parent)
        {
            TextView text = new TextView(ValuesResTest.this);
            Resources res = ValuesResTest.this.getResources();
            // 使用尺度资源来设置文本框的高度、宽度
            text.setWidth((int) res.getDimension(R.dimen.cell_width));
            text.setHeight((int) res.getDimension(R.dimen.cell_height));
            // 使用字符串资源设置文本框的内容
            text.setText(textIds[position]);
            // 使用颜色资源来设置文本框的背景色
            text.setBackgroundResource(colorIds[position]);
            text.setTextSize(20);
            text.setTextSize(getResources()
                .getInteger(R.integer.font_size));
            return text;
        }
    };
    GridView grid = (GridView)findViewById(R.id.grid01);
    // 为 GridView 设置 Adapter
    grid.setAdapter(ba);
}
}
```

上面的程序中粗体字代码分别使用了前面定义的字符串资源、数组资源和颜色资源，运行上面的程序，将可以看到如图 6.1 所示的界面。

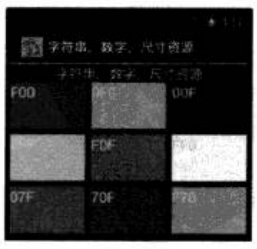


图 6.1　使用字符串、颜色、尺寸资源

与定义字符串资源类似的是，Android 也允许使用资源文件来定义 boolean 常量，例如在/res/values 目录下增加一个 bools.xml 文件，该文件的根元素也是<resources.../>，根元素内通过<bool.../>子

元素来定义 boolean 常量，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <bool name="is_male">true</bool>
    <bool name="is_big">false</bool>
</resources>
```

一旦在资源文件中定义了如上所示的资源文件之后，接下来在 Java 代码中按如下语法格式访问即可：

```
[<package_name>.]R.bool.bool_name
```

在 XML 文件中按如下格式即可访问资源：

```
@[<package_name>:]bool/bool_name
```

例如为了在 Java 代码中获取指定 boolean 变量的值，可通过如下代码来实现：

```
Resources res = getResources();
boolean is_male = res.getBoolean(R.bool.is_male);
```

与定义字符串资源类似的是，Android 也允许使用资源文件来定义整型常量，例如在/res/values 目录下增加一个 integers.xml 文件（文件名可以自由选择），该文件的根元素也是<resources.../>，根元素内通过<integer.../>子元素来定义整型常量，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <integer name="my_size">32</integer>
   <integer name="book_numbers">12</integer>
</resources>
```

一旦在资源文件中定义了如上所示的资源文件之后，接下来在 Java 代码中按如下语法格式访问即可：

```
[<package_name>.]R.integer.integer_name
```

在 XML 文件中按如下格式即可访问资源：

```
@[<package_name>:]integer/integer_name
```

例如为了在 Java 代码中获取指定整型变量的值，可通过如下代码来实现：

```
Resources res = getResources();
int my_size = res.getInteger(R.bool.my_size);
```

6.3 数组（Array）资源

上面的程序中在 Java 代码中定义了两个数组，Android 并不推荐在 Java 代码中定义数组，因为 Android 允许通过资源文件来定义数组资源。

Android 采用位于/res/values 目录下的 arrays.xml 文件来定义数组，定义数组时 XML 资源文件的根元素也是<resources.../>元素，该元素内可包含如下三种子元素。

- <array..../>子元素：定义普通类型的数组。例如 Drawable 数组。
- <string-array..../>子元素：定义字符串数组。
- <integer-array..../>子元素：定义整数数组。

一旦在资源文件中定义了数组资源，接下来就可以在 Java 文件中通过如下形式来访问资源了：

```
[<package_name>.]R.array.array_name
```

在 XML 代码中则可通过如下形式进行访问：

```
@[<package_name>:]array/array_name
```

为了能在 Java 程序访问到实际数组，Resources 提供了如下方法。

- String[] getStringArray(int id)：根据资源文件中字符串数组资源的名称来获取实际的字符串数组。
- int[] getIntArray(int id)：根据资源文件中整型数组资源的名称来获取实际的整型数组。
- TypedArray obtainTypedArray(int id)：根据资源文件中普通数组资源的名称来获取实际的普通数组。

TypedArray 代表一个通用类型的数组，该类提供了 getXxx(int index)方法来获取指定索引处的数组元素。

下面为该应用程序增加如下数组资源文件。

程序清单：codes\06\6.3\ArrayResTest\res\values\arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 定义一个 Drawable 数组 -->
    <array name="plain_arr">
        <item>@color/c1</item>
        <item>@color/c2</item>
        <item>@color/c3</item>
        <item>@color/c4</item>
        <item>@color/c5</item>
        <item>@color/c6</item>
        <item>@color/c7</item>
        <item>@color/c8</item>
        <item>@color/c9</item>
    </array>
    <!-- 定义字符串数组 -->
    <string-array name="string_arr">
        <item>@string/c1</item>
        <item>@string/c2</item>
        <item>@string/c3</item>
        <item>@string/c4</item>
        <item>@string/c5</item>
        <item>@string/c6</item>
        <item>@string/c7</item>
        <item>@string/c8</item>
        <item>@string/c9</item>
    </string-array>
    <!-- 定义字符串数组 -->
    <string-array name="books">
        <item>疯狂 Java 讲义</item>
        <item>疯狂 Ajax 讲义</item>
        <item>疯狂 Android 讲义</item>
    </string-array>
</resources>
```

定义了上面的数组资源之后，既可在 XML 文件中使用这些数组资源，也可在 Java 程序

中使用这些数组资源。例如如下界面布局文件中定义了一个 ListView 数组，并将 android:entries 属性值指定为一个数组。界面布局文件代码如下。

程序清单：codes\06\6.3\ArrayResTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center_horizontal"
      >
<!-- 省略其他组件定义 -->
...
<!-- 定义 ListView 组件，使用了数组资源 -->
<ListView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:entries="@array/books"
      />
</LinearLayout>
```

接下来程序中无须定义数组，程序直接使用资源文件中定义的数组。程序代码如下。

程序清单：codes\06\6.3\ArrayResTest\src\org\crazyit\res\ArrayResTest.java

```
public class ArrayResTest extends Activity
{
      // 获取系统定义的数组资源
      String[] texts;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            texts = getResources().getStringArray(R.array.string_arr);
            // 创建一个 BaseAdapter 对象
            BaseAdapter ba = new BaseAdapter()
            {
                  @Override
                  public int getCount()
                  {
                        // 指定一共包含 9 个选项
                        return texts.length;
                  }
                  @Override
                  public Object getItem(int position)
                  {
                        // 返回指定位置的文本
                        return texts[position];
                  }
                  @Override
                  public long getItemId(int position)
                  {
                        return position;
                  }
                  // 重写该方法，该方法返回的 View 将作为的 GridView 的每个格子
                  @Override
                  public View getView(int position
                              , View convertView, ViewGroup parent)
                  {
                        TextView text = new TextView(ArrayResTest.this);
```

```
                Resources res = ArrayResTest.this.getResources();
                // 使用尺度资源来设置文本框的高度、宽度
                text.setWidth((int) res.getDimension(R.dimen.cell_width));
                text.setHeight((int) res.getDimension(R.dimen.cell_height));
                // 使用字符串资源设置文本框的内容
                text.setText(texts[position]);
                TypedArray icons = res.obtainTypedArray(R.array.plain_arr);
                // 使用颜色资源来设置文本框的背景色
                text.setBackgroundDrawable(icons.getDrawable(position));
                text.setTextSize(20);
                return text;
            }
        };
        GridView grid = (GridView) findViewById(R.id.grid01);
        // 为 GridView 设置 Adapter
        grid.setAdapter(ba);
    }
}
```

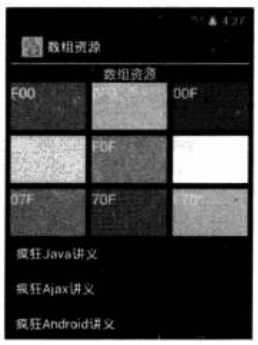


图 6.2　使用数组资源

上面的程序中粗体字代码就是使用数组资源的关键代码。运行上面的程序将看到如图 6.2 所示的结果。

6.4　使用（Drawable）资源

Drawable 资源是 Android 应用中使用最广泛的资源，也是 Android 应用中最灵活的资源，它不仅可以直接使用*.png、*.jpg、*.gif、*.9.png 等图片作为资源，也可使用多种 XML 文件作为资源。只要一份 XML 文件可以被系统编译成 Drawable 子类的对象，那么这份 XML 文件即可作为 Drawable 资源。

Drawable 资源通常保存在/res/drawable 目录，实际上可能保存在/res/drawable-ldpi、/res/drawable- mdpi、/res/drawable-hdpi 目录下。

下面详细介绍各种 Drawable 资源。

6.4.1　图片资源

图片资源是最简单的 Drawable 资源，只要把*.png、*.jpg、*.gif 等格式的图片放入/res/drawable-xxx 目录下，Android SDK 就会在编译应用中自动加载该图片，并在 R 资源清单类中生成该资源的索引。

注意：

Android 要求图片资源的文件名必须符合 Java 的标识符的命名规则，否则 Android SDK 无法为该图片在 R 类中生成资源索引。



一旦系统在 R 资源清单类中生成了指定资源的索引，接下来就可以在 Java 类中使用如下语法格式来访问该资源：

```
[<package_name>.]R.drawable.<file_name>
```

在 XML 代码中则按如下语法格式来访问该资源：

```
@[<package_name>:]drawable/file_name
```

为了在程序中获得实际的 Drawable 对象，Resources 提供了 Drawable getDrawable(int id)方法，该方法即可根据 Drawable 资源在 R 清单类中的 ID 来获取实际的 Drawable 对象。

关于图片资源的示例，本书前面已有大量介绍，故此处不再赘述。

6.4.2 StateListDrawable 资源

StateListDrawable 用于组织多个 Drawable 对象。当使用 StateListDrawable 作为目标组件的背景、前景图片时，StateListDrawable 对象所显示的 Drawable 对象会随目标组件状态的改变而自动切换。

定义 StateListDrawable 对象的 XML 文件的根元素为<selector.../>，该元素可以包含多个<item.../>元素，该元素可指定如下属性。

- android:color 或 android:drawable：指定颜色或 Drawable 对象。
- android:state_xxx：指定一个特定状态。

例如如下语法格式：

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >
    <!-- 指定特定状态下的颜色 -->
    <item android:color="hex_color"
        android:state_pressed=["true" | "false"] />
</selector>
```

StateListDrawable 的<item.../>元素所支持的状态有如表 6.3 所示的几种。

表 6.3 StateListDrawable 支持的状态

属 性 值	含 义
android:state_active	代表是否处于激活状态
android:state_checkable	代表是否处于可勾选状态
android:state_checked	代表是否处于已勾选状态
android:state_enabled	代表是否处于可用状态
android:state_first	代表是否处于开始状态
android:state_focused	代表是否处于已得到焦点状态
android:state_last	代表是否处于结束状态
android:state_middle	代表是否处于中间状态
android:state_pressed	代表是否处于已被按下状态
android:state_selected	代表是否处于已被选中状态
android:state_window_focused	代表是否窗口已得到焦点状态

实例：高亮显示正在输入的文本框

前面知道，使用 EditText 时可指定一个 android:textColor 属性，该属性用于指定文本框的文字颜色。前面介绍该属性时总是直接给它一个颜色值，因此该文本框的文字颜色总是固定的。借助于 StateListDrawable 对象，可以让文本框的文字颜色随文本框的状态动态改变。

为本系统提供如下 Drawable 资源文件。

程序清单：codes\06\6.4\StateListDrawableTest\res\drawable-mdpi\my_image.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- 指定获得焦点时的颜色 -->
    <item android:state_focused="true"
        android:color="#f44"/>
    <!-- 指定失去焦点时的颜色 -->
    <item android:state_focused="false"
        android:color="#eee"/>
</selector>
```

上面的资源文件中指定了目标组件得到焦点、失去焦点时使用不同的颜色，接下来可以在定义 EditText 时使用该资源。

程序清单：codes\06\6.4\StateListDrawableTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 使用 StateListDrawable 资源 -->
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="@drawable/my_image"
    />
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="@drawable/my_image"
    />
</LinearLayout>
```

该应用的 Java 程序代码不需任何修改，只要显示该界面布局即可。运行该程序将看到如图 6.3 所示的界面。



图 6.3　输入文本时高亮显示

通过使用 StateListDrawable 不仅可以让文本框里文字的颜色随文本框状态的改变而切换，也可让按钮的背景图片随按钮状态的改变而切换，实际上 StateListDrawable 的功能非常灵活，它可以让各种组件的背景、前景随状态的改变而切换。

6.4.3　LayerDrawable 资源

与 StateListDrawable 有点类似，LayerDrawable 也可包含一个 Drawable 数组，因此系统将会按这些 Drawable 对象的数组顺序来绘制它们，索引最大的 Drawable 对象将会被绘制在最上面。

定义 LayerDrawable 对象的 XML 文件的根元素为<layer-list.../>，该元素可以包含多个<item.../>元素，该元素可指定如下属性。

- android:drawable：指定作为 LayerDrawable 元素之一的 Drawable 对象。
- android:id：为该 Drawable 对象指定一个标识。
- android:buttom|top|left|button：它们用于指定一个长度值，用于指定将该 Drawable 对象绘制到目标组件的指定位置。

例如如下语法格式：

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
      <!-- 指定一个 Drawable 元素 -->
      <item android:id="@android:id/background"
            android:drawable="@drawable/grow" />
</layer-list >
```

实例：定制拖动条的外观

前面知道，使用 SeekBar 时可指定一个 android:progressDrawable 属性，该属性可改变 SeekBar 的外观，借助于 LayerDrawable 即可改变 SeekBar 的规定、已完成部分的 Drawable 对象。

例如定义如下 Drawable 资源：

程序清单：codes\06\6.4\LayerDrawableTest\res\drawable-mdpi\my_bar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
      <!-- 定义轨道的背景 -->
      <item android:id="@android:id/background"
            android:drawable="@drawable/grow" />
      <!-- 定义轨道上已完成部分的外观-->
      <item android:id="@android:id/progress"
            android:drawable="@drawable/ok" />
</layer-list>
```

除此之外，该示例还定义了如下 LayerDrawable 对象。

程序清单：codes\06\6.4\LayerDrawableTest\res\drawable-mdpi\layout_logo.xml

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
      <bitmap android:src="@drawable/ic_launcher"
        android:gravity="center" />
    </item>
    <item android:top="25dp" android:left="25dp">
      <bitmap android:src="@drawable/ic_launcher"
        android:gravity="center" />
    </item>
    <item android:top="50dp" android:left="50dp">
      <bitmap android:src="@drawable/ic_launcher"
        android:gravity="center" />
    </item>
</layer-list>
```

上面的程序中定义了三个“层叠”在一起的 Drawable 对象，接着在界面布局使用上面的 my_bar.xml 定义的 Drawable 对象来改变 SeekBar 的外观，并通过 ImageView 来显示上面 layout_logo 的 Drawble 组件。界面布局的代码片段如下。

程序清单：codes\06\6.4\LayerDrawableTest\res\layout\main.xml

```
<!-- 定义一个拖动条，并改变轨道外观 -->
<SeekBar
      android:id="@+id/bar"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:max="100"
      android:progressDrawable="@drawable/my_bar"
```

```
        />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/layout_logo"
        />
```

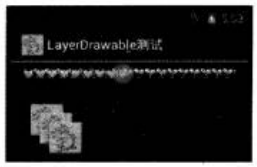


图 6.4　使用 LayerDrawable

该程序的代码无须任何改变，直接加载、显示上面的界面布局文件即可，运行该程序出现如图 6.4 所示的界面。

6.4.4　ShapeDrawable 资源

ShapeDrawable 用于定义一个基本的几何图形（如矩形、圆形、线条等），定义 ShapeDrawable 的 XML 文件的根元素是<shape.../>元素，该元素可指定如下属性。

- android:shape=["rectangle" | "oval" | "line" | "ring"]：指定定义哪种类型的几何图形。

定义 ShapeDrawable 对象的完整语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <!-- 定义几何图形的四个角的弧度 -->
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />
    <!-- 定义使用渐变色填充 -->
    <gradient
        android:angle="integer"
        android:centerX="integer"
        android:centerY="integer"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=["linear" | "radial" | "sweep"]
        android:usesLevel=["true" | "false"] />
    <!-- 定义几何形状的内边距 -->
    <padding
        android:left="integer"
        android:top="integer"
        android:right="integer"
        android:bottom="integer" />
    <!-- 定义几何形状的大小 -->
    <size
        android:width="integer"
        android:color="color"
        android:dashWidth="integer"
        android:dashGap="integer" />
    <!-- 定义使用单种颜色填充 -->
    <solid
        android:color="color" />
    <!-- 定义为几何形状绘制边框 -->
    <stroke
        android:width="integer"
        android:color="color"
```

```
            android:dashWidth="integer"
            android:dashGap="integer" />
</shape>
```

下面通过示例来介绍 ShapeDrawable 资源的定义和使用。

实例：椭圆形、渐变背景的文本框

前面介绍 TextView 时知道该组件可指定一个 android:background 属性，该属性用于为该文本框指定背景。大部分时候，文本框的背景只是一个简单的图片，或者只是一个简单的颜色。

如果程序使用 ShapeDrawable 资源作为文本框的 android:background 属性，则可以在 Android 应用中做出各种外观的文本框。下面先定义如下的 ShapeDrawable 资源。

程序清单：codes\06\6.4\ShapeDrawableTest\res\drawable-mdpi\my_shape_1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="rectangle">
      <!-- 设置填充颜色 -->
      <solid android:color="#fff"/>
      <!-- 设置四周的内边距 -->
      <padding android:left="7dp"
            android:top="7dp"
            android:right="7dp"
            android:bottom="7dp" />
      <!-- 设置边框 -->
      <stroke android:width="3dip" android:color="#ff0" />
</shape>
```

接下来定义如下 ShapeDrawable 资源。

程序清单：codes\06\6.4\ShapeDrawableTest\res\drawable-mdpi\my_shape_2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="rectangle">
      <!-- 定义填充渐变颜色 -->
      <gradient
            android:startColor="#FFFF0000"
            android:endColor="#80FF00FF"
            android:angle="45"/>
      <!-- 设置内填充 -->
      <padding android:left="7dp"
            android:top="7dp"
            android:right="7dp"
            android:bottom="7dp" />
      <!-- 设置圆角矩形 -->
      <corners android:radius="8dp" />
</shape>
```

再定义如下 ShapeDrawable 资源。

程序清单：codes\06\6.4\ShapeDrawableTest\res\drawable-mdpi\my_shape_3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="oval">
```

```
        <!-- 定义填充渐变颜色 -->
        <gradient
              android:startColor="#ff0"
              android:endColor="#00f"
              android:angle="45"
              android:type="sweep"/>
        <!-- 设置内填充 -->
        <padding android:left="7dp"
              android:top="7dp"
              android:right="7dp"
              android:bottom="7dp" />
        <!-- 设置圆角矩形 -->
        <corners android:radius="8dp" />
    </shape>
```

定义了上面三个 ShapeDrawable 资源之后，接下来在界面布局文件中用这三个 ShapeDrawable 资源作为文本框的背景。界面布局文件代码如下。

程序清单：codes\06\6.4\ShapeDrawableTest\res\drawable-mdpi\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="@drawable/my_shape_1"
      />
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="@drawable/my_shape_2"
      />
<EditText
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="@drawable/my_shape_3"
      />
</LinearLayout>
```



图 6.5　使用 ShapeDrawable

使用 Activity 加载、显示上面的界面布局文件，将可以看到如图 6.5 所示的界面。

6.4.5　ClipDrawable 资源

ClipDrawable 代表从其他位图上截取的一个“图片片段”。在 XML 文件中定义 ClipDrawable 对象使用<clip.../>元素，该元素的语法为：

```
<?xml version="1.0" encoding="utf-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
      android:drawable="@drawable/drawable_resource"
      android:clipOrientation=["horizontal" | "vertical"]
      android:gravity=["top" | "bottom" | "left" | "right" | "center_vertical" |
            "fill_vertical" | "center_horizontal" | "fill_horizontal" |
            "center" | "fill" | "clip_vertical" | "clip_horizontal"] />
```

上面的语法格式中可指定如下三个属性。

- android:drawable：指定截取的源 Drawable 对象。
- android:clipOrientation：指定截取方向，可设置水平截取或垂直截取。
- android:gravity：指定截取时的对齐方式。

使用 ClipDrawable 对象时可调用 setLevel(int level)方法来设置截取的区域大小，当 level 为 0 时，截取的图片片段为空；当 level 为 10000 时，截取整张图片。

下面以一个示例来说明 ClipDrawble 对象的用法。

实例：徐徐展开的风景

因为 ClipDrawble 对象可调用 setLevel(int level)控制截取图片的部分，因此本示例只要设置一个定时器，让程序不断调用 ClipDrawble 的 setLevel(int level)方法即可实现图片徐徐展开的效果。

程序先定义如下 ClipDrawble 对象。

程序清单：codes\06\6.4\ClipDrawableTest\res\drawable-mdpi\my_clip.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<clip xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/shuangta"
    android:clipOrientation="horizontal"
    android:gravity="center">
</clip>
```

上面的程序控制从中间开始截取图片，截取方向为水平截取。接下来程序将通过一个定时器来定期修改 ClipDrawable 对象的 level，即可实现图片徐徐张开的效果。

程序清单：codes\06\6.4\ClipDrawableTest\src\org\crazyit\res\ClipDrawableTest.java

```
public class ClipDrawableTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ImageView imageview = (ImageView) findViewById(R.id.image);
        // 获取图片所显示的 ClipDrawable 对象
        final ClipDrawable drawable = (ClipDrawable)
            imageview.getDrawable();
        final Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 如果该消息是本程序所发送的
                if (msg.what == 0x1233)
                {
                    // 修改 ClipDrawable 的 level 值
                    drawable.setLevel(drawable.getLevel() + 200);
                }
            }
        };
        final Timer timer = new Timer();
        timer.schedule(new TimerTask()
        {
            @Override
```

```
                public void run()
                {
                    Message msg = new Message();
                    msg.what = 0x1233;
                    // 发送消息，通知应用修改 ClipDrawable 对象的 level 值。
                    handler.sendMessage(msg);
                    // 取消定时器
                    if (drawable.getLevel() >= 10000)
                    {
                        timer.cancel();
                    }
                }
            }, 0, 300);
        }
    }
```

运行上面的程序，将看到如图 6.6 所示的结果。

从图 6.6 所示的运行结果可以看出，通过使用这种徐徐展开的图片，用户会感觉就像进度条一样——实际上，实际应用中完全可以用这种 ClipDrawable 对象来实现图片进度条。

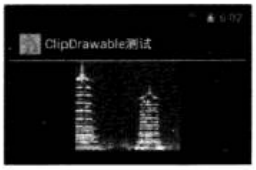


图 6.6　使用 ClipDrawable 对象

6.4.6　AnimationDrawable 资源

AnimationDrawable 代表一个动画，关于 Android 动画的知识本书后面还有更详细的介绍，本节只是先介绍一下如何定义 AnimationDrawable 资源。Android 既支持传统的逐帧动画（类似于电影方式，一张图片、一张图片地切换），也支持通过平移、变换计算出来的补间动画。

下面以补间动画为例来介绍如何定义 AnimationDrawable 资源，定义补间动画的 XML 资源文件以<set.../>元素作为根元素，该元素内可以指定如下 4 个元素：

- alpha：设置透明度的改变。
- scale：设置图片进行缩放改变。
- translate：设置图片进行位移变换。
- rotate：设置图片进行旋转。

定义动画的 XML 资源应该放在/res/anmi 路径下，当使用 ADT 创建一个 Android 应用时，默认不会包含该路径，开发者需要自行创建该路径。

定义补间动画的思路很简单：设置一张图片的开始状态（包括透明度、位置、缩放比、旋转度）、并设置该图片的结束状态（包括透明度、位置、缩放比、旋转度），再设置动画的持续时间，Android 系统会使用动画效果把这张图片从开始状态变换到结束状态。

设置补间动画的语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@[package:]anim/interpolator_resource"
    android:shareInterpolator=["true" | "false"]
    android:duration="持续时间">
    <alpha
        android:fromAlpha="float"
        android:toAlpha="float" />
    <scale
        android:fromXScale="float"
        android:toXScale="float"
        android:fromYScale="float"
        android:toYScale="float"
```

```
            android:pivotX="float"
            android:pivotY="float" />
        <translate
            android:fromXDelta="float"
            android:toXDelta="float"
            android:fromYDelta="float"
            android:toYDelta="float" />
        <rotate
            android:fromDegrees="float"
            android:toDegrees="float"
            android:pivotX="float"
            android:pivotY="float" />
</set>
```

上面语法格式中包含了大量的 fromXxx、toXxx 属性，这些属性就用于定义图片的开始状态、结束状态。除此之外，当进行缩放变换（scale）、旋转（rotate）变换时，还需要指定如下 pivotX、pivotY 两个属性，这两个属性用于指定变换的“中心点”——比如进行旋转变换时，需要指定“旋轴点”；进行缩放变换时，需要指定“中心点”。

除此之外，上面<set.../>、<alpha.../>、<scale.../>、<translate.../>、<rotate.../>都可指定一个 android:interpolator 属性，该属性指定动画的变化速度，可以实现匀速、正加速、负加速、无规则变加速等，Android 系统的 R.anim 类中包含了大致常量，它们定义了不同的动画速度，例如：

- linear_interpolator：匀速变换。
- accelerate_interpolator：加速变换。
- decelerate_interpolator：减速变换。

如果程序想让<set.../>元素下所有的变换效果使用相同的动画速度，则可指定 android:shareInterpolator="true"。

例如下面的资源文件定义了一个动画资源。

程序清单：codes\06\6.4\AnimationDrawable\res\anim\my_anim.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator"
    android:duration="5000">
    <!-- 定义缩放变换 -->
    <scale android:fromXScale="1.0"
        android:toXScale="1.4"
        android:fromYScale="1.0"
        android:toYScale="0.6"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="true"
        android:duration="2000"
    />
    <!-- 定义位移变换 -->
    <translate android:fromXDelta="10"
        android:toXDelta="130"
        android:fromYDelta="30"
        android:toYDelta="-80"
        android:duration="2000"
    />
</set>
```

上面的动画资源文件十分简单，它只指定了图片资源需要进行两种变换：缩放变换和位

移变换。

一旦定义了上面的动画资源文件，接下来就可以在 XML 文件中按如下语法格式来访问它：

```
@[<package_name>:]anim/file_name
```

在 Java 代码中则按如下语法格式来访问它：

```
[<package>.]R.anim.<file_name>
```

为了在 Java 代码中获取实际的 Animation 对象，则可调用 AnimationUtils 的如下方法：

- loadAnimation(Context ctx, int resId)

下面的程序示范了如何使用 AnimationDrawable 资源。

程序清单：codes\06\6.4\AnimationDrawable\src\org\crazyit\res\AnimationDrawable.java

```
public class AnimationDrawable extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        final ImageView image = (ImageView)findViewById(R.id.image);
        // 加载动画资源
        final Animation anim = AnimationUtils.loadAnimation(this,
            R.anim.my_anim);
        // 设置动画结束后保留结束状态
        anim.setFillAfter(true);
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 开始动画
                image.startAnimation(anim);
            }
        });
    }
}
```



图 6.7　使用 AnimationDrawable

上面的程序使用了界面布局中的两个组件：一个 ImageView、一个 Button，这是两个最普通的组件，故此处不再给出界面布局的代码。运行上面的程序，将看到如图 6.7 所示的动画效果。

上面的程序中①号代码设置动画结束后保留图片的变换结果。本来 Android 的 API 文档中说明可以在<alpha.../>、<scale.../>、<translate.../>、<rotate.../>等元素中指定 android:fillAfter 为 true 来实现这个效果，但实际上要为<set.../>设置 android:fillAfter 为 true 才可以。

6.5　属性动画（Property Animation）资源

Animator 代表一个属性动画，但它只是一个抽象类，通常会使用它的子类：AnimatorSet、

ValueAnimator、ObjectAnimator、TimeAnimator。关于Android动画的知识，本书后面还有更详细的介绍，本节只是先介绍一下如何定义属性动画资源。

定义属性动画的XML资源文件能以如下三个元素中的任意一个作为根元素。

- <set.../>：它是一个父元素，用于包含其他<objectAnimator.../>、<animator.../>或<set.../>子元素，该元素定义的资源代表AnimatorSet对象。
- <objectAnimator.../>：用于定义ObjectAnimator动画。
- <animator.../>：用于定义ValueAnimator动画。

定义属性动画的语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<set android:ordering=["together" | "sequentially"]>
    <objectAnimator
        android:propertyName="string"
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:interpolator=""
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>
    <animator
        android:duration="int"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color"
        android:startOffset="int"
        android:repeatCount="int"
        android:interpolator=""
        android:repeatMode=["repeat" | "reverse"]
        android:valueType=["intType" | "floatType"]/>
    <set>
        ...
    </set>
</set>
```

实例：不断渐变的背景色

该实例将会使用属性动画来控制组件背景色不断渐变。该实例所使用的属性动画资源文件如下。

程序清单：codes\06\6.5\AnimatorTest\res\animator\color_anim.xml

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator xmlns:android="http://schemas.android.com/apk/res/android"
    android:propertyName="backgroundColor"
    android:duration="3000"
    android:valueFrom="#FF8080"
    android:valueTo="#8080FF"
    android:repeatCount="infinite"
    android:repeatMode="reverse"
    android:valueType="intType">
</objectAnimator>
```

上面的代码定义了一个ObjectAnimator对象，接下来程序就可以通过属性动画来控制指定组件背景色不断改变。下面是该实例的Activity代码。

程序清单：codes\06\6.5\AnimatorTest\src\org\crazyit\res\AnimatorTest.java

```
public class AnimatorTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout container = (LinearLayout)
            findViewById(R.id.container);
        // 添加 MyAnimationView 组件
        container.addView(new MyAnimationView(this));
    }
    public class MyAnimationView extends View
    {
        public MyAnimationView(Context context)
        {
            super(context);
            // 加载动画资源
            ObjectAnimator colorAnim = (ObjectAnimator) AnimatorInflater
                .loadAnimator(AnimatorTest.this, R.animator.color_anim);
            colorAnim.setEvaluator(new ArgbEvaluator());
            // 对该 View 本身应用属性动画
            colorAnim.setTarget(this);
            // 开始指定动画
            colorAnim.start();
        }
    }
}
```



图 6.8　不断渐变的背景色

上面的程序中粗体字代码使用 AnimatorInflater 工具类加载了指定动画资源文件、将该动画资源文件转换为 ObjectAnimator 对象。接下来程序对 MyAnimationView 本身应用该动画，将可以看到该组件的背景色不断变化，如图 6.8 所示。

6.6　使用原始 XML 资源

在某些时候，Android 应用有一些初始化的配置信息、应用相关的数据资源需要保存，一般推荐使用 XML 文件来保存它们，这种资源就被称为原始 XML 资源。下面介绍如何定义、获取原始 XML 资源。

➤➤6.6.1　定义原始 XML 资源

原始 XML 资源一般保存在/res/xml 路径下——当使用 ADT 创建 Android 应用时，/res 目录下并没有包含 xml 目录，开发者应该自行手动创建 xml 目录。

接下来 Android 应用对原始 XML 资源没有任何特殊的要求，只要它是一份格式良好的 XML 文档即可。

一旦成功地定义了原始XML资源，接下来在XML文件中可通过如下语法格式来访问它：

```
@[<package_name>:]xml/file_name
```

在 Java 代码中则按如下语法格式来访问它：

```
[<package_name>.]R.xml。<file_name>
```

为了在 Java 程序中获取实际的 XML 文档，可以通过 Resources 的如下两个方法来获取。

- XmlResourceParser getXml(int id)：获取 XML 文档，并使用一个 XmlPullParser 来解析该 XML 文档，该方法返回一个解析器对象（XmlResourceParser 是 XmlPullParser 的子类）。
- InputStream openRawResource(int id)：获取 XML 文档对应的输入流。

大部分时候，我们可以直接调用 getXml(int id)方法来获取 XML 文档，并对该文档进行解析。Android 默认使用内置的 Pull 解析器来解析 XML 文件。

除了 Pull 解析之外，Java 开发者还可使用 DOM 或 SAX 对 XML 文档进行解析。一般的 Java 应用会使用 JAXP API 来解析 XML 文档。对于实际的 Java EE 项目而言，使用 JDOM 或 dom4j 进行解析可能更加简便。



提示：

Pull 解析器是一个开源项目，既可以用于 Android 应用，也可以用于 Java EE 应用。如果需要在 Java EE 应用中使用 Pull 解析器，则需要自行下载并添加 Pull 解析器的 JAR 包。不过 Android 平台已经内置了 Pull 解析器，而且 Android 系统本身也使用 Pull 解析器来解析各种 XML 文档，因此 Android 推荐开发者 Pull 解析器来解析 XML 文档。

Pull 解析方式有点类似于 SAX 解析，它们都采用事件驱动方式来进行解析。当 Pull 解析器开始解析之后，开发者可不断地调用 Pull 解析器的 next()方法获取下一个解析事件（开始文档、结束文档、开始标签、结束标签等），当处于某个元素处时，可调用 XmlPullParser 的 getAttributeValue()方法来获取该元素的属性值，也可调用 XmlPullParser 的 nextText()方法来获取文本节点的值。

如果开发者希望使用 DOM、SAX 或其他解析器来解析 XML 资源，那么可调用 openRawResource (int id)方法来获取 XML 资源对应的输入流，这样即可自行选择解析器来解析指该 XML 资源了。



提示：

使用其他 XML 解析器需要开发者自行下载并安装解析器的 JAR 包。关于 DOM、SAX、JAXP、dom4j、JDOM 的相关知识，或者读者需要获取更多关于 XML 的的知识，请参考疯狂 Java 体系的《疯狂 XML 讲义》。

6.6.2 使用原始 XML 文件

下面为示例程序添加一个原始的 XML 文件，将该 XML 文件放到/res/xml 目录下，该 XML 文件的内容很简单。XML 资源的内容如下。

程序清单：codes\06\6.6\XmlResTest\res\xml\books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
    <book price="99.0" 出版日期="2008 年">疯狂 Java 讲义</book>
    <book price="89.0" 出版日期="2009 年">轻量级 Java EE 企业应用实战</book>
    <book price="69.0" 出版日期="2009 年">疯狂 Ajax 讲义</book>
</books>
```

接下来就可以在 Java 程序中获取该 XML 资源，并解析该 XML 资源中的信息。Java 程

序如下。

程序清单：codes\06\6.6\XmlResTest\src\org\crazyit\res\XmlResTest.java

```
public class XmlResTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 bn 按钮，并为该按钮绑定事件监听器
        Button bn = (Button) findViewById(R.id.bn);
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 根据 XML 资源的 ID 获取解析该资源的解析器
                // XmlResourceParser 是 XmlPullParser 的子类
                XmlResourceParser xrp = getResources().getXml(R.xml.books);
                try
                {
                    StringBuilder sb = new StringBuilder("");
                    // 还没有到 XML 文档的结尾处
                    while (xrp.getEventType()
                        != XmlResourceParser.END_DOCUMENT)
                    {
                        // 如果遇到了开始标签
                        if (xrp.getEventType() == XmlResourceParser.
                        START_TAG)
                        {
                            // 获取该标签的标签名
                            String tagName = xrp.getName();
                            // 如果遇到 book 标签
                            if (tagName.equals("book"))
                            {
                                // 根据属性名来获取属性值
                                String bookName = xrp.getAttribute-
                            Value(null,
                                        "price");
                                sb.append("价格：");
                                sb.append(bookName);
                                // 根据属性索引来获取属性值
                                String bookPrice = xrp.getAttribute-
                            Value(1);
                                sb.append("      出版日期：");
                                sb.append(bookPrice);
                                sb.append(" 书名：");
                                // 获取文本节点的值
                                sb.append(xrp.nextText());
                            }
                            sb.append("\n");
                        }
                        // 获取解析器的下一个事件
                        xrp.next(); //①
                    }
                    EditText show = (EditText) findViewById(R.id.show);
                    show.setText(sb.toString());
                }
                catch (XmlPullParserException e)
```

```
                {
                    e.printStackTrace();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

上面的程序中①号粗体字代码用于不断获取 Pull 解析的解析事件，程序中第一行粗体字只要解析事件不等于 XmlResourceParser.END_ DOCUMENT（也就是还没有解析结束），程序将一直解析下去，通过这种方式即可把整份 XML 文档的内容解析出来。

上面的程序中包含一个按钮和一个文本框，当用户单击该按钮时，程序将会解析指定 XML 文档，并把文档中的内容显示出来。运行该程序，然后单击“解析 XML 资源”按钮，程序显示如图 6.9 所示的界面。



图 6.9 使用 XML 资源

6.7 使用布局（Layout）资源

实际上从我们学习第一个 Android 应用开始，已经开始接触 Android 的 Layout 资源了，因此此处不会详细介绍 Android Layout 资源的知识，会对 Layout 资源进行简单的归纳。

Layout 资源文件应放在/res/layout 目录下，Layout 资源文件的根元素通常是各种布局管理器，比如 LinearLayout、TableLayout、FrameLayout 等，接着在该布局管理器中定义各种 View 组件即可。

一旦在 Android 项目中定义了 Layout 资源，接下来在 XML 文件中可通过如下语法格式来访问它：

```
@[<package_name>:]layout/file_name
```

在 Java 代码中则按如下语法格式来访问它：

```
[<package_name>.]R.layout.<file_name>
```

6.8 使用菜单（Menu）资源

前面已经介绍过 Android 的菜单支持，前面介绍菜单时分别介绍了如何使用 Java 代码来实现菜单和使用 XML 资源文件定义菜单。

实际上 Android 推荐使用 XML 资源文件来定义菜单，使用 XML 资源文件定义菜单将会提供更好的解耦。由于前面介绍介绍过如何使用 XML 资源文件定义菜单，因此此处不再详细介绍菜单资源文件的内容，只是对其进行简单的归纳。

Android 菜单资源文件放在/res/menu 目录下，菜单资源的根元素通常是<menu.../>元素，<menu.../>元素无须指定任何属性。

一旦在 Android 项目中定义了 Layout 资源，接下来在 XML 文件中可通过如下语法格式

来访问它：

```
@[<package_name>:]menu/file_name
```

在 Java 代码中则按如下语法格式来访问它：

```
[<package_name>.]R.menu.<file_name>
```

6.9 样式（Style）和主题（Theme）资源

样式和主题资源都是用于对 Android 应用进行“美化”的，只要充分利用 Android 应用的样式和主题资源，开发者可以开发出各种风格的 Android 应用。

6.9.1 样式资源

如果我们经常需要对某个类型的组件指定大致相似的格式，比如字体、颜色、背景色等，如果每次都要为 View 组件重复指定这些属性，无疑会有大量的工作量，而且不利于项目后期的维护。

类似于 Word，Word 也提供了样式来管理格式：一个样式等于一组格式的集合，如果设置某段文本使用某个样式，那么该样式的所有格式将会整体应用于这段文本。Android 的样式与此类似，Android 样式也包含一组格式，为一个组件设置使用某个样式时，该样式所包含的全部格式将会应用于该组件。



提示：

一个样式相当于多个格式的集合，其他 UI 组件通过 style 属性来指定样式，这就相当于把该样式包含的所有格式同时应用于该 UI 组件。

Android 的样式资源文件也放在/res/values 目录下，样式资源文件的根元素是<resources.../>元素，该元素内可包含多个<style.../>子元素，每个<style.../>元素定义一个样式。<style.../>元素指定如下两个属性。

- name：指定样式的名称。
- parent：指定该样式所继承的父样式。当继承某个父样式时，该样式将会获得父样式中定义的全部格式。当然，当前样式也可以覆盖父样式中指定的格式。

<style.../>元素内可包含多个<item.../>子元素，每个<item.../>子元素定义一个格式项。例如为应用定义如下样式文件。

程序清单：codes\06\6.9\StyleResTest\res\values\my_style.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <!-- 定义一个样式，指定字体大小、字体颜色 -->
    <style name="style1">
        <item name="android:textSize">20sp</item>
        <item name="android:textColor">#00d</item>
    </style>
    <!-- 定义一个样式，继承前一个颜色 -->
    <style name="style2" parent="@style/style1">
        <item name="android:background">#ee6</item>
        <item name="android:padding">8dp</item>
```

```
        <!-- 覆盖父样式中指定的属性 -->
        <item name="android:textColor">#000</item>
    </style>
</resources>
```

上面的样式资源中定义了两个样式，其中第二个样式继承了第一个样式，而且第二个样式中的 textColor 属性覆盖了父样式中的 textColor 属性。

一旦定义了上面的样式资源之后，接下来就可以在 XML 资源中按如下语法格式来使用样式了：

```
@[<package_name>:]style/file_name
```

下面是该示例中的界面布局文件，该布局文件中包含两个文本框，这两个文本框分别使用两个样式。

程序清单：codes\06\6.9\StyleResTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 指定使用 style1 的样式 -->
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/style1"
    style="@style/style1"
    />
<!-- 指定使用 style2 的样式 -->
<EditText
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/style2"
    style="@style/style2"
    />
</LinearLayout>
```

上面的界面布局文件中并未为两个文本框指定任何格式，只是为它们分别指定了使用 style1、style2 的样式，这两个样式包含的格式就会应用到这两个文本框。运行上面的程序，将看到如图 6.10 所示的界面。

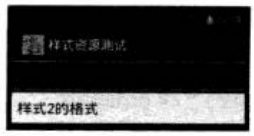


图 6.10 使用 Style 资源

6.9.2 主题资源

与样式资源非常相似，主题资源的 XML 文件通常也放在/res/values 目录下，主题资源的 XML 文档同样以<resource.../>元素作为根元素，同样使用<style.../>元素来定义主题。

主题与样式的区别主要体现在：

- 主题不能作用于单个的 View 组件，主体应该对整个应用中的所有 Activity 起作用，或对指定的 Activity 起作用。
- 主题定义的格式应该是改变窗口外观的格式，例如窗口标题、窗口边框等。

实例：给所有窗口添加边框、背景

下面通过一个示例来介绍主题的用法。为了给所有窗口都添加边框、背景，先在/res/values/my_ style.xml 文件中增加一个主题，定义主体的<style.../>片段如下：

```
<style name="CrazyTheme">
      <item name="android:windowNoTitle">true</item>
      <item name="android:windowFullscreen">true</item>
      <item name="android:windowFrame">@drawable/window_border</item>
      <item name="android:windowBackground">@drawable/star</item>
</style>
```

上面的主题定义中使用了两个 Drawable 资源，其中@drawable/star 是一张图片；@drawable/window_border 是一个 ShapeDrawable 资源，该资源对应的 XML 文件代码如下。

程序清单：codes\06\6.8\StyleResTest\res\drawable_mdpi\window_border.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
      android:shape="rectangle">
      <!-- 设置填充颜色 -->
      <solid android:color="#0fff"/>
      <!-- 设置四周的内边距 -->
      <padding android:left="7dp"
            android:top="7dp"
            android:right="7dp"
            android:bottom="7dp" />
      <!-- 设置边框 -->
      <stroke android:width="10dip" android:color="#f00" />
</shape>
```

定义了上面主题之后，接下来即可在 Java 代码中使用该主题，例如如下代码：

```
public void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      setTheme(R.style.CrazyTheme);
      setContentView(R.layout.linear_layout_3);
}
```

大部分时候，在 AndroidManifest.xml 文件中对指定应用、指定 Activity 应用主题更加简单。如果我们想让应用中全部窗口使用该主题，只要为<application.../>元素添加 android:theme 属性。属性值是一个主题的名字，如以下代码所示：

```
<application android:theme="@style/CrazyitTheme">
...
</application>
```

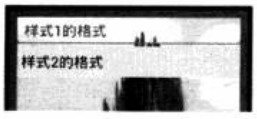


图 6.11 使用主题资源

如果你只是想让程序中的某个 Activity 拥有这个主题，那么你可以修改<activity.../>元素，同样通过 android:theme 指定主题即可。

本应用在 AndroidManifest.xml 文件的<application.../>元素中添加了 android:theme= "@style/CrazyTheme"属性，运行程序可看到如图 6.11 所示的界面。

从图 6.11 所示的效果可以看出，该窗口没有标题，窗口背景也被改变了，窗口全屏显示……这些都是自定义主题控制的。



提示： 可能会有读者觉得窗口边框弄得这么粗，显得很难看，其实读者可以自行控制。笔者之所以弄得这么粗，是为了让读者看到窗口边框的效果。

Android 中提供了几种内置的主题资源，这些主题通过查询 Android.R.style 类可以看到。例如前面介绍的对话框风格的窗口，我们只要采用如下代码来定义某个 Activity 即可。

```
<activity android:theme="@android:style/Theme.Dialog">
...
</activtiy>
```

与样式类似的是，Android 主题同样支持继承。如果开发过程中还想利用某个主题，但需要对它进行局部修改，则可通过继承系统主题来实现自定义主题。例如如下代码片段：

```
<style name="CrazyTheme" parent="@android:style/Theme.Dialog">
...
</activtiy>
```

上面定义的 CrazyTheme 主题继承了 android.R.style.Theme.Dialog 主题，那么接下来在该<style.../>元素中添加的<item.../>子元素就可覆盖系统主题的部分属性了。

6.10 属性（Attribute）资源

前面已经介绍过自定义 View 组件的开发，自定义 View 组件与 Android 系统提供的 View 组件一样，既可在 Java 代码中使用，也可在 XML 界面布局代码中使用。

当在 XML 布局文件中使用 Android 系统提供的 View 组件时，开发者可以指定多个属性，这些属性可以很好地控制 View 组件的外观行为。如果用户开发的自定义 View 组件也需要指定属性，就需要属性资源的帮助了。

属性资源文件也放在/res/values 目录下，属性资源文件的根元素也是<resources.../>，该元素里包含如下两个子元素。

- attr 子元素：定义一个属性。
- declare-styleable 子元素：定义一个 styleable 对象，每个 styleable 对象就是一组 attr 属性的集合。

当我们使用属性文件定义了属性之后，接下来就可以在自定义组件的构造器中通过 AttributeSet 对象来获取这些属性了。

例如我们想开发一个默认带动画效果的图片，该图片显示时，自动从全透明变到完全不透明，为此我们需要开发一个自定义组件，但这个自定义组件需要指定一个额外 duration 属性，该属性控制动画的持续时间。

为了在自定义组件中使用 duration 属性，需要先定义如下属性资源文件。

程序清单：codes\06\6.10\AttrResTest\res\values\attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- 定义一个属性 -->
    <attr name="duration">
    </attr>
    <!-- 定义一个 styleable 对象来组合多个属性 -->
    <declare-styleable name="AlphaImageView">
```

```
        <attr name="duration"/>
    </declare-styleable>
</resources>
```

上面的属性资源文件定义了该属性之后，至于到底在哪个 View 组件中使用该属性，该属性到底能发挥什么作用，就不归属性资源文件管了。属性资源所定义的属性到底可以发挥什么作用，取决于自定义组件的代码实现。

例如如下自定义的 AlphaImageView，它获取了定义该组件所指定的 duration 属性之后，根据该属性来控制图片的透明度的改变。程序代码如下。



提示：

在属性资源中定义<declare-styleable.../>元素时，也可在其内部直接使用<attr.../>定义属性，使用<attr.../>时指定一个 format 属性即可，例如上面我们可以指定<attr name="duration" format="integer"/>。

程序清单：codes\06\6.10\AttrResTest\src\org\crazyit\res\AlphaImageView.java

```
public class AlphaImageView extends ImageView
{
    // 图像透明度每次改变的大小
    private int alphaDelta = 0;
    // 记录图片当前的透明度。
    private int curAlpha = 0;
    // 每隔多少毫秒透明度改变一次
    private final int SPEED = 300;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                // 每次增加 curAlpha 的值
                curAlpha += alphaDelta;
                if (curAlpha > 255) curAlpha = 255;
                // 修改该 ImageView 的透明度
                AlphaImageView.this.setAlpha(curAlpha);
            }
        }
    };
    public AlphaImageView(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        TypedArray typedArray = context.obtainStyledAttributes(attrs,
                R.styleable.AlphaImageView);
        // 获取 duration 参数
        int duration = typedArray
                .getInt(R.styleable.AlphaImageView_duration, 0);
        // 计算图像透明度每次改变的大小
        alphaDelta = 255 * SPEED / duration;
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        this.setAlpha(curAlpha);
        super.onDraw(canvas);
        final Timer timer = new Timer();
```

```
            // 按固定间隔发送消息，通知系统改变图片的透明度
            timer.schedule(new TimerTask()
            {
                @Override
                public void run()
                {
                    Message msg = new Message();
                    msg.what = 0x123;
                    if (curAlpha >= 255)
                    {
                        timer.cancel();
                    }
                    else
                    {
                        handler.sendMessage(msg);
                    }
                }
            }, 0, SPEED);
        }
    }
```

上面的程序中粗体字代码用于获取定义 AlphaImageView 时指定的 duration 属性，并根据该属性计算图片的透明度的变化幅度，接着程序重写了 ImageView 的 onDraw(Canvas canvas) 方法，该方法启动了一个任务调度来控制图片透明度的改变。

上面的粗体字代码中的 R.styleable.AlphaImageView、R.styleable.AlphaImageView_duration 都是 Android SDK 根据属性资源文件自动生成的。

接着在界面布局文件使用 AlphaImageView 时可以为它指定一个 duration 属性，注意该属性位于“http://schemas.android.com/apk/res/ + 项目子包”命名空间下，例如本应用的包名为 org.crazyit.res，那么 duration 属性就位于“http://schemas.android.com/apk/res/org.crazyit.res”命名空间下。

下面是该应用的界面布局文件的代码。

程序清单：codes\06\6.10\AttrResTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:crazyit="http://schemas.android.com/apk/res/org.crazyit.res"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 使用自定义组件，并指定属性资源中定义的属性 -->
<org.crazyit.res.AlphaImageView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/ee"
    crazyit:duration="60000"
    />
</LinearLayout>
```

上面的程序中第一行粗体字代码用于导入 http://schemas.android.com/apk/res/org.crazyit.res 命名空间，并指定该命名空间对应的短名前缀为 crazyit；程序第二行粗体字代码用于为 AlphaImageView 组件指定自定义属性 duration 的属性值为 60000。

主程序无须做任何特殊的控制，只要简单地加载并显示上面的界面布局文件，运行该程序时即可看到该图片从无到有，慢慢显示出来的效果。

6.11 使用原始资源

除了上面介绍的各种 XML 文件、图片文件之外，Android 应用可能还需要用到大量其他类型的资源，比如声音资源等。实际上，声音对于 Android 应用非常重要，选择合适的音效可以让 Android 应用增色不少。

类似声音文件及其他各种类型的文件，只要 Android 没有为之提供专门的支持，这种资源都被称为原始资源。Android 的原始资源可以放在如下两个地方。

- 位于/res/raw 目录下，Android SDK 会处理该目录下原始资源，Android SDK 会在 R 清单类中为该目录下的资源生成一个索引项。
- 位于/assets/目录下，该目录下的资源是更彻底的原始资源。Android 应用需要通过 AssetManager 来管理该目录下的原始资源。

Android SDK 会为位于/res/raw/目录下的资源在 R 类中生成一个索引项，接下来在 XML 文件中可通过如下语法格式来访问它：

```
@[<package_name>:]raw/file_name
```

在 Java 代码中则按如下语法格式来访问它：

```
[<package_name>.]R.raw.<file_name>
```

通过上面的索引项，Android 应用就可以非常方便地访问/res/raw 目录下的原始资源，至于获取原始资源之后如何处理，则完全取决于实际项目的需要。

AssetManager 是一个专门管理/assets/目录下原始资源的管理器类，AssetManager 提供了如下两个常用方法来访问 Assets 资源。

- InputStream open(String fileName)：根据文件名来获取原始资源对应的输入流。
- AssetFileDescriptor openFd(String fileName)：根据文件名来获取原始资源对应的 AssetFileDescriptor。AssetFileDescriptor 代表了一项原始资源的描述，应用程序可通过 AssetFileDescriptor 来获取原始资源。

下面的程序示范了如何使用声音，先在应用的/res/raw/目录下放入一个 bomb.mp3 文件——Android SDK 会自动处理该目录下的资源，会在 R 清单类中为它生成一个索引项：R.raw.bomb。

接下来我们再往/assets/目录下放入一个 shot.mp3 文件——需要通过 AssetManager 进行管理。

下面的程序中定义两个按钮，一个按钮用于播放/res/raw/目录下的声音文件，另一个用于播放/assets/目录下的声音文件。程序界面布局代码很简单，此处不再给出。下面是程序代码。

程序清单：codes\06\6.11\RawResTest\src\org\crazyit\res\RawResTest.java

```
public class RawResTest extends Activity
{
    MediaPlayer mediaPlayer1 = null;
    MediaPlayer mediaPlayer2 = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 直接根据声音文件的 ID 来创建 MediaPlayer
```

```
            mediaPlayer1 = MediaPlayer.create(this, R.raw.bomb);
            // 获取该应用的AssetManager
            AssetManager am = getAssets();
            try
            {
                // 获取指定文件对应的AssetFileDescriptor
                AssetFileDescriptor afd = am.openFd("shot.mp3");
                mediaPlayer2 = new MediaPlayer();
                // 使用MediaPlayer加载指定的声音文件
                mediaPlayer2.setDataSource(afd.getFileDescriptor());
                mediaPlayer2.prepare();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
            // 获取第一个按钮，并为它绑定事件监听器
            Button playRaw = (Button) findViewById(R.id.playRaw);
            playRaw.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View arg0)
                {
                    // 播放声音
                    mediaPlayer1.start();
                }
            });
            // 获取第二个按钮，并为它绑定事件监听器
            Button playAsset = (Button) findViewById(R.id.playAsset);
            playAsset.setOnClickListener(new OnClickListener()
            {
                @Override
                public void onClick(View arg0)
                {
                    // 播放声音
                    mediaPlayer2.start();
                }
            });
        }
    }
```

上面的程序中第一行粗体字代码用于获取/res/raw/目录下原始资源文件；第二段粗体字代码则利用了AssetManager来获取/assets/目录下的原始资源文件。

上面的程序中利用了MediaPlayer来播放声音，MediaPlayer是Android提供的一个播放声音的类，本书后面还有关于该类更详细的介绍。

6.12 国际化和资源自适应

全球化的Internet需要全球化的软件。全球化软件即意味着同一种版本的产品能够容易地适用于不同地区的市场。引入国际化的目的是为了提供自适应、更友好的用户界面，并不需要改变程序的逻辑功能。国际化的英文单词是Internationalization，因为这个单词太长了，有时也简称I18N，其中I是这个单词的第一个字母，18表示中间省略的字母个数，而N代表这个单词的最后一个字母。

一个国际化支持很好的应用，会随着在不同区域使用而呈现出本地语言的提示。这个过

程也被称为 Localization，即本地化。类似于国际化可以称为 I18N，本地化也可以称为 L10N。

Android 所采用到资源管理方式可以非常方便地实现程序国际化。

6.12.1 Java 国际化的思路

Java 程序的国际化的思路是将程序中的标签、提示等信息放在资源文件中，程序需要支持哪些国家、语言环境，就需要提供相应的资源文件。资源文件是 key-value 对，每个资源文件中的 key 是不变的，但 value 则随不同国家、语言改变。图 6.12 显示了 Java 程序国际化的解决思路。

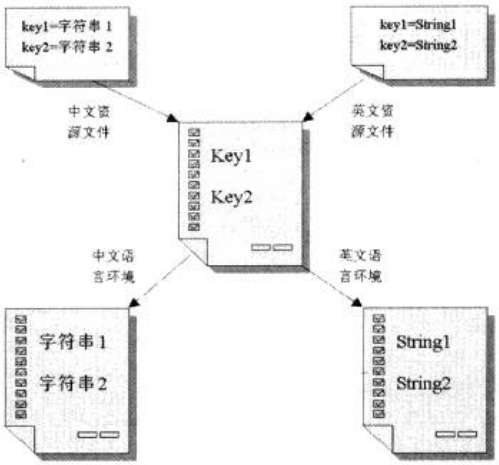


图 6.12　Java 程序国际化的思路

Java 程序的国际化主要通过如下三个类完成。

- java.util.ResourceBundle：用于加载一个国家、语言资源包。
- java.util.Locale：用于封装一个特定的国家/区域、语言环境。
- java.text.MessageFormat：用于格式化带占位符的字符串。

为了实现程序的国际化，必须先提供程序所需要的资源文件。资源文件的内容适合很多 key-value 对。其中 key 是程序使用的部分，而 value 则是程序界面的显示字符串。

资源文件的命名可以有如下三种形式。

- baseName _ language _country.properties
- baseName _language.properties
- baseName.properties

其中 baseName 是资源文件的基本名，用户可以自由定义。而 language 和 country 都不可随意变化，必须是 Java 所支持的语言和国家。

6.12.2 Java 支持的语言和国家

事实上，Java 不可能支持所有的国家和语言，如需要获取 Java 所支持的语言和国家，可调用 Locale 类的 getAvailableLocales 方法获取，该方法返回一个 Locale 数组，该数组里包含了 Java 所支持的语言和国家。

下面的程序简单地示范了如何获取 Java 所支持的国家和语言。

程序清单：codes\06\6.12\LocaleList.java

```
public class LocaleList
{
    public static void main(String[] args)
    {
        // 返回 Java 所支持的全部国家和语言的数组
        Locale[] localeList = Locale.getAvailableLocales();
        // 遍历数组的每个元素，依次获取所支持的国家和语言
        for (int i = 0; i < localeList.length ; i++ )
        {
            // 打印出所支持的国家和语言
            System.out.println(localeList[i].getDisplayCountry()
                + "=" + localeList[i].getCountry()
                + " " + localeList[i].getDisplayLanguage()
                + "=" + localeList[i].getLanguage());
        }
    }
}
```

程序的运行结果如图 6.13 所示。



图 6.13 Java 国际化所支持的语言和国家

通过该程序，我们就可以获得 Java 程序所支持的国家/语言环境。



提示： 虽然可以通过查阅相关资料来获取 Java 语言所支持的国家/语言环境，但如果这些资料不是随手可得的，则可以通过上面的程序来获得 Java 语言所支持的国家/语言环境。

6.12.3 完成程序国际化

对于如下最简单的程序：

```
public class RawHello
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```

这个程序的执行结果也很简单，肯定是打印出简单的“Hello World”字符串，不管在哪里执行都不会有任何改变！为了让该程序支持国际化，则肯定不能在程序直接输出“Hello

World”的字符串，这种写法直接输出一个字符串常量，永远不会有任何改变。为了让程序可以输出不同的字符串，此处绝不可使用该字符串常量。

为了让上面输出的字符串常量可以改变，我们将需要输出的各种字符串（不同国家/语言环境对应不同的字符串）定义在资源包中。

我们为上面程序提供如下两个文件。

第一个文件：mess_zh_CN.properties，该文件的内容为：

```
#资源文件的内容是 key-value 对。
hello=您好!
```

第二个文件：mess_en_US.properties，该文件的内容为：

```
#资源文件的内容是 key-value 对。
hello=Welcome You!
```

对于包含非西欧字符的资源文件，Java 提供了一个工具来处理该文件：native2ascii，这个工具可以在%JAVA_HOME%/bin 路径下找到。使用该工具的语法格式如下：

```
native2ascii 源资源文件 目的资源文件
```

如果我们在命令窗口输入如下指令：

```
#使用 native2ascii 命令处理 mess_zh_CN.properties 文件，生成 aa.properties 文件
native2ascii mess_zh_CN.properties aa.properties
```

上面的命令将生成一个 aa.properties 文件，该文件才是我们需要的资源文件，该文件看上去包含很多乱码，其实是非西欧字符的 Unicode 编码方式，这完全正常。将该文件重命名为 mess_zh_CN.properties 即可。

我们看到这两份文件文件名的 baseName 是相同的：mess。前面已经介绍了资源文件的三种命名方式，其中 baseName 后面的国家、语言必须是 Java 所支持的国家、语言组合。

将上面的 Java 程序修改成如下形式。

程序清单：codes\06\6.12\Hello.java

```
public class Hello
{
    public static void main(String[] args)
    {
        // 取得系统默认的国家/语言环境
        Locale myLocale = Locale.getDefault(Locale.Category.FORMAT);
        // 根据指定国家/语言环境加载资源文件
        ResourceBundle bundle = ResourceBundle
            .getBundle("mess" , myLocale);
        // 打印从资源文件中取得的消息
        System.out.println(bundle.getString("hello"));
    }
}
```

上面的程序中的打印语句不再是直接打印“Hello World”字符串，而是打印了从资源包中读取的信息。如果在中文环境下运行该程序，将打印“您好!”；如果我们在“控制面板”将机器的语言环境设置成美国，然后再次运行该程序，将打印“Welcome You!”字符串。

通过上面的简单程序，我们可以体会到 Java 程序的国际化是多么简单！

从上面的程序可以看出，如果我们希望程序完成国际化，只需要将不同国家/语言（Locale）的提示信息分别以不同文件存放。例如简体中文的语言资源文件就是 Xxx_zh_CN.

properties 文件，而美国英语的语言资源文件就是 Xxx_en_US.properties 文件。

Java 程序国际化的关键类是 ResourceBundle，它有一个静态方法：getBundle(String baseName , Locale locale);。该方法将根据 Locale 加载资源文件，而 Locale 封装了一个国家、语言，例如简体中文的环境可以用简体中文的 Locale 代表，美国英语的环境可以用美国英语的 Locale 代表。

从上面的资源文件的命名中可以看出，不同语言、国家环境的资源文件的 baseName 是相同的，即 baseName 为 mess 的资源文件有很多个，不同国家、语言环境对应不同的资源文件。

例如通过如下代码来加载资源文件：

```
// 根据指定国家/语言环境加载资源文件
ResourceBundle bundle = ResourceBundle.getBundle("mess" , myLocale);
```

上面的代码将会加载 baseName 为 mess 的系列资源文件的其中之一，到底加载其中的哪一个，则取决于 myLocale，对于简体中文的 Locale，则加载 mess_zh_CN.properties 文件。

一旦加载了该文件，该资源文件的内容就是多个 key-value 对，程序就根据 key 来获取指定信息，例如获取 key 为 hello 的消息，该消息是“您好！”——这就是 Java 程序国际化的过程。

对于美国英语的 Locale，则加载 mess_en_US.properties，该文件中的 key 为 hello 的消息是“Welcome You!”。

Java程序国际化的关键类是ResourceBundle和Locale，ResourceBundle来根据不同Loacle加载语言资源文件，再根据指定 key 取得已加载语言资源文件中的字符串即可。

6.12.4 为 Android 应用提供国际化资源

Android 程序的国际化资源更加方便——因为 Android 本身就采用了 XML 资源文件来管理所有字符串消息，只要为各消息提供不同语言、国家对应的内容即可。

通过前面的介绍我们知道，Android 应用使用 res\values 目录下的资源文件来保存程序中用到的字符串消息，为了给这些消息提供不同语言、国家的版本，开发者需要为 values 目录添加几个不同的语言国家版本。不同 values 文件夹的命名方式为：

```
values-语言代码-r 国家代码
```

例如希望下面的应用支持简体中文、美式英语两种环境，则需要在 res 目录下添加 values-zh-rCN、values-en-rUS 两个目录。

如果希望应用程序的图片也能随语言、国家环境改变，那么还需要为 drawable 目录添加几个不同的语言国家版本。不同 drawable 文件夹的命名方式为：

```
drawale-语言代码-r 国家代码
```



提示： 如果还需要为 drawable 目录按分辨率提供文件夹，则可以在后面追加分辨率后缀，例如 drawable-zh-rCN-mdpi、drawable-zh-rCN-hdpi、drawable-zh-rCN-xhdpi、drawable-en-rUS-mdpi、drawable-en-rUS-hdpi、drawable-en-rUS-xhdpi 等。

下面的程序分别为 values 目录、drawable 目录创建了简体中文、美式英语两个版本，如图 6.14 所示。



图 6.14　国际化资源文件

在 res\drawable-zh-rCN-mdpi、res\drawable-en-rUS-mdpi 目录下分别添加 logo.png 图片，这两个图片并不相同，一个是简体中文环境的图片，一个是美式英语环境的图片。

在 res\values-en-rUS 、 res\values-zh-rCN 目录下分别创建 strings.xml 文件，很明显，该文件中存放的就是字符串资源。

其中 res\values-en-rUS 目录下的 strings.xml 是美式英语的字符串资源文件，而 res\values-zh-rCN 目录下的 strings.xml 是简体中文的字符串资源文件。其中 res\values-en-rUS 目录下的 strings.xml 文件内容如下：

```
<resources>
      <string name="ok">OK</string>
      <string name="cancel">Cancel</string>
      <string name="msg">Hello , Android!</string>
</resources>
```

res\values-zh-rCN 目录下的 strings.xml 文件内容如下：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <string name="ok">确定</string>
      <string name="cancel">取消</string>
      <string name="msg">你好啊，可爱的小机器人！</string>
</resources>
```

正如 Java 国际化中看到的，不同语言的国际化资源文件中所有消息的 key 是相同的，不同语言、国家环境下，消息资源 key 对应的 value 不同。

由于 Android 的消息资源本身就是采用 XML 文件来提供的，所以不需要额外的处理。接下来就可以在 Android 应用中使用这些国际化消息资源了。

➤➤6.12.5　国际化 Android 应用

Android 的设计本身就是国际化的，当开发者在 XML 界面布局文件中、在 Java 代码中加载字符串资源时，Android 的国际化机制已经起作用了。

对于下面的界面布局文件：

程序清单：codes\06\6.12\i18n\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      >
<TextView
      android:id="@+id/show"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:lines="2"
      android:gravity="top"
      />
<ImageView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:src="@drawable/logo"
```

```
        />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center_horizontal"
        >
    <!-- 两个按钮的文本都是通过消息资源指定的 -->
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ok"
        />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cancel"
        />
    </LinearLayout>
    </LinearLayout>
```

上面三行粗体字代码与前面的界面布局代码没有任何改变，但它本身就没有把字符串内容“写死”在布局文件中，它本身就是去加载资源文件中的字符串值，此时 Android 的国际化机制就可发挥作用了——如果 Android 是简体中文的环境，就加载 res\values-zh-rCN\strings.xml 文件中的字符串资源、加载 res\drawable-zh-rCN 目录下的 Drawable 资源；如果是美式英语的环境，就加载 res\values-en-rUS\strings.xml 文件中的字符串资源、加载 res\drawable-en-rUS 目录下的 Drawable 资源

与此类似的是，在 Java 代码中编程时，程序同样可以根据资源 ID 设置字符串内容，而不是以硬编码的方式设置为固定的字符串内容。例如如下程序代码：

程序清单：codes\06\6.12\i18n\src\org\crazyit\res\I18NTest.java

```
public class I18NTest extends Activity
{
    TextView tvShow;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tvShow = (TextView) findViewById(R.id.show);
        // 设置文本框所显示的文本
        tvShow.setText(R.string.msg);
    }
}
```

上面的文件为 EditText 设置文本内容时并未以硬编码的方式设置为固定的字符串内容，而是设置为消息资源的 name——相当于国际化消息的 key。类似的，Android 系统的国际化机制同样可以发挥作用：如果 Android 是简体中文的环境，就加载 res\values-zh-rCN\strings.xml 文件中的字符串资源；如果是美式英语的环境，就加载 res\values-en-rUS\strings.xml 文件中的字符串资源。

将手机设为美式英语环境（通过 Android 系统的 Settings→Language&keyboard settings→Select language→English(United States)进行设置），运行该程序即可看到如图 6.15 所示的界面。

将手机设为简体中文环境（通过 Android 系统的 Settings→Language&keyboard settings→Select language→中文(简体)进行设置），运行该程序即可看到如图 6.16 所示的界面。



图 6.15　美式英语的环境



图 6.16　简体中文的环境

从图 6.15、图 6.16 可以看出，两个程序是同一个程序，只是由于它们的运行环境不同，Android 系统会控制程序加载不同的资源文件，因此程序界面上的图片、字符串就完全不同了。



提示：

上面程序中的标题并未改变，这是因为这个标题对应的字符串资源只有一份，而且是保存在 values 目录下的，因此不管是简体中文环境，还是美式英语环境，系统总是加载这份资源文件，因此程序的标题是固定的。如果需要对程序标题也进行国际化，不难，只要为程序标题对应的字符串消息名（app_name）分别提供美式英语、简体中文的消息资源即可。

6.13　自适应不同屏幕的资源

开发 Android 应用有一个比较烦人的地方是：Android 设备的屏幕尺寸、分辨率差别非常大，而开发者开发的 Android 应用总希望能在所有 Android 设备上运行，因此开发 Android 应用就需要考虑不同屏幕的适应性问题。



提示：

相比之下，开发 iOS 应用要更简单，因为 iOS 只有手机和平板电脑两种设备，它们的屏幕尺寸、分辨率都是固定的，因此需要考虑的设备更少。

前面已经提到，Android 默认把 drawable 目录（存放图片等 Drawable 资源的目录）分为 drawable-ldpi、drawable-mdpi、drawable-hdpi、drawable-xhdpi 这 4 个子目录，这 4 个子目录就是为不同分辨率准备的图片。

通常来说，屏幕资源需要考虑如下两个方面。

- 屏幕尺寸：屏幕尺寸可分为 **small**（小屏幕）、**normal**（中等屏幕）、**large**（大屏幕）、**xlarge**（超大屏幕）**4** 种。
- 屏幕分辨率：屏幕分辨率可分为 **ldpi**（低分辨率）、**mdpi**（中等分辨率）、**hdpi**（高分辨率）、**xhdpi**（超高分辨率）**4** 种。
- 屏幕方向：屏幕方向可分为：**land**（横屏）和 **port**（竖屏）两种。

图 6.17 显示了不同屏幕尺寸、不同分辨率对应的通用说法。

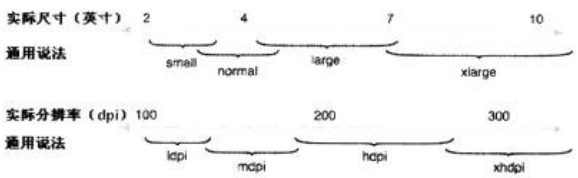


图 6.17　不同尺寸、不同分辨率的屏幕

当我们为不同尺寸的屏幕设置用户界面时，每种用户界面总有一个最低的屏幕尺寸要求（低于该屏幕尺寸就没法运行），上面这些通用说法中屏幕尺寸总有一个最低分辨率，该最低分辨率是以 dp 为单位 的，因此我们在定义界面布局时应该尽量考虑使用 dp 为单位。

下面是上面 4 种屏幕尺寸所需的最低分辨率：

- xlarge 屏幕尺寸至少需要 960dp x 720dp。
- large 屏幕尺寸至少需要 640dp x 480dp
- normal 屏幕尺寸至少需要 470dp x 320dp
- small 屏幕尺寸至少需要 426dp x 320dp

通过上面的介绍不难发现，为了提供自适应不同屏幕的资源，最简单的做法就是为不同屏幕尺寸、不同屏幕分辨率提供相应的布局资源、Drawable 资源。

- 屏幕分辨率：可以为 drawable 目录增加如下后缀：ldpi（低分辨率）、mdpi（中等分辨率）、hdpi（高分辨率）、xhdpi（超高分辨率）、nodpi（默认分辨率），分别为不同分辨率的屏幕提供资源。
- 屏幕尺寸：可以为 layout、values 等目录增加如下后缀：small、normal、large 和 xlarge，分别为不同尺寸的屏幕提供相应资源。

注意：

从 Android 3.2 开始，Android 建议直接使用真实的屏幕尺寸来定义屏幕尺寸。Android 3.2 支持在 layout、values 目录后添加 sw<N>dp（屏幕尺寸至少宽 N 个 dp 才能使用该资源）、w<N>dp（屏幕尺寸可用宽度为 N 个 dp 可使用该资源）、h<N>dp（屏幕尺寸可用高度为 N 个 dp 才能使用该资源）。例如可指定 layout-sw600dp，表明该设备屏幕的宽度大于或等于 600 个 dp 时使用该目录下的布局资源。



- 屏幕方向：可以为 layout、values 等目录增加后缀 land 和 port，分别为横屏、竖屏提供相应的资源。

下面的示例在 layout 目录下定义了一个界面布局文件。

程序清单：codes\06\6.13\DpiTest\res\layout\main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:src="@drawable/a"/>
</RelativeLayout>
```

上面的界面布局文件指定显示@drawable/a 对应的图片资源，在 mdpi 设备上运行程序时（比如 3.2 英寸屏幕、320×480 分辨率），系统将会使用/res/drawable-mdpi 目录下的图片。运行该程序，将看到如图 6.18 所示界面。

在 mdpi 设备上运行程序时（比如 4.0 英寸屏幕、480*800 分辨率），系统将会使用/res/drawable-hdpi 目录下的图片。运行该程序，将看到如图 6.19 所示界面。



图 6.18　mdpi 屏幕使用 drawable-mdpi 目录下图片



图 6.19　hdpi 屏幕使用 drawable-hdpi 目录下图片

下面的示例提供了两份布局文件，其中一份放在 layout-normal 目录下，一份放在 layout-large 目录下。其中 layout-normal 目录下的布局文件代码如下。

程序清单：codes\06\6.13\ScreenSizeTest\res\layout-normal\main.xml

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
      <Button
         android:text="第一个按钮"
           android:layout_width="match_parent"
           android:layout_height="wrap_content" />
      <Button
         android:text="第二个按钮"
           android:layout_width="match_parent"
           android:layout_height="wrap_content" />
      <Button
         android:text="第三个按钮"
           android:layout_width="match_parent"
           android:layout_height="wrap_content" />
</LinearLayout>
```

layout-large 目录下的布局文件代码如下。

程序清单：codes\06\6.13\ScreenSizeTest\res\layout-large\main.xml

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
      <Button
         android:text="第一个按钮"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content" />
      <Button
         android:text="第二个按钮"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content" />
      <Button
         android:text="第三个按钮"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content" />
</LinearLayout>
```

在普通尺寸的屏幕（3.2 英寸屏幕，分辨率为 320×480）上运行该程序，系统将会加载 layout-normal 目录下的 main.xml 布局文件。运行该程序，将看到如图 6.20 所示界面。

在大尺寸的屏幕（7.0 英寸屏幕，分辨率为 1024×600）上运行该程序，系统将会加载

layout-large 目录下的 main.xml 布局文件。运行该程序，将看到如图 6.21 所示界面。



图 6.20 正常尺寸的屏幕加载 layout-normal 目录下的界面布局文件



图 6.21 大尺寸的屏幕加载 layout-large 目录下的界面布局文件

6.14 本章小结

本章主要介绍了 Android 应用资源的相关内容。Android 应用资源是一种非常优秀的、高解耦设计，通过使用资源文件，Android 应用可以把各种字符串、图片、颜色、界面布局等交给 XML 文件配置管理，这样就避免在 Java 代码中以硬编码的方式直接定义这些内容。学习本章需要掌握 Android 应用资源的存储方式、Android 应用资源的使用方式。除此之外，Android 应用的字符串资源、颜色资源、尺寸资源、数组资源、图片资源、各种 Drawable 资源、原始 XML 资源、布局资源、菜单资源、样式和主题资源、属性资源、原始资源各种资源文件都需要重点掌握。

本章最后还介绍了 Android 应用的国际化，由于 Android 应用的国际化实际上是以 Java 国际化为基础的，因此本章还简单介绍了 Java 程序的国际化，这些内容都需要读者认真掌握。

CHAPTER

7

第7章 图形与图像处理

本章要点

- Android 的图形处理基础
- Bitmap 与 BitmapFactory
- 继承 View 来 Android 中绘图
- 掌握 Canvas、Paint、Path 等绘图 API
- 双缓冲机制
- 使用 Matrix 对图像进行几何变换
- 通过 drawBitmapMesh 方法扭曲图像
- 使用不同的 Shader 类渲染图形
- 逐帧动画
- 补间动画
- 属性动画
- 开发自定义补间动画
- SurfaceView 的绘图机制
- 继承 SurfaceView 开发动画

正如前面介绍的，决定 Android 应用是否被用户接受的重要方面就是用户界面，为了提供友好的用户界面，就需要在应用中使用图片了。Android 系统提供了丰富的图片功能支持，包括处理静态图片和动画等。

Android 系统提供了 ImageView 显示普通静态图片，也提供了 AnimationDrawable 来开发逐帧动画，还可通过 Animation 对普通图片使用补减动画。图形、图像处理不仅是 Android 系统的应用界面非常重要，而且 Android 系统上益智类游戏、2D 游戏都需要大量的图形、图像处理。所谓游戏，本质就是提供更逼真的、能模拟某种环境的用户界面，并根据某种规则来响应用户操作。为了提供更逼真的用户界面，需要借助于图形处理。

学习本章内容之后，读者应该能熟练掌握 Android 系统的图形、图像处理，这样就可在 Android 平台上开发出俄罗斯方块、五子棋等小游戏。本章将会通过弹球游戏、飞行游戏雏型来帮助读者掌握 Android 2D 游戏开发的入门知识。

7.1 使用简单图片

前面的 Android 应用中已经大量使用了简单图片，图片不仅可以使用 ImageView 来显示，也可作为 Button、Window 的背景。从广义的角度来看，Android 应用中的图片不仅包括*.png、*.jpg、*.gif 等各种格式的位图，也包括使用 XML 资源文件定义的各种 Drawable 对象。

7.1.1 使用 Drawable 对象

为 Android 应用增加了 Drawable 资源之后，Android SDK 会为这份资源在 R 清单文件中创建一个索引项：R.drawable.file_name。

接下来既可在 XML 资源文件中通过@drawable/file_name 来访问该 Drawable 对象，也可在 Java 代码中通过 R.drawable.file_name 访问该 Drawable 对象。

需要指出的是，R.drawable.file_name 是一个 int 类型的常量，它只代表 Drawable 对象的 ID，如果 Java 程序中需要获取实际的 Drawable 对象，则可调用 Resources 的 getDrawable(int id)方法来获取。

由于前面已经介绍了大量关于 Drawable 的示例，故此处不再给出示例。

7.1.2 Bitmap 和 BitmapFactory

Bitmap 代表一张位图，BitmapDrawable 里封装的图片就是一个 Bitmap 对象。开发者为了把一个 Bitmap 对象包装成 BitmapDrawable 对象，可以调用 BitmapDrawable 的构造器：

```
// 把一个 Bitmap 对象包装成 BitmapDrawable 对象
BitmapDrawable drawable = new BitmapDrawable(bitmap);
```

如果需要获取 BitmapDrawable 所包装的 Bitmap 对象，则可调用 BitmapDrawable 的 getBitmap()方法，如下面的代码所示：

```
// 获取一个 BitmapDrawable 所包装的 Bitmap 对象。
Bitmap bitmap = drawable.getBitmap();
```

除此之外，Bitmap 还提供了一些静态方法来创建新的 Bitmap 对象，例如如下常用方法。

- createBitmap(Bitmap source, int x, int y, int width, int height)：从源位图 source 的

指定坐标点（给定 x、y）开始，从中“挖取”宽 width、高 height 的一块出来，创建新的 Bitmap 对象。

- createScaledBitmap(Bitmap src, int dstWidth, int dstHeight, boolean filter)：对源位图 src 进行缩放，缩放成宽 dstWidth、高 dstHeight 的新位图。
- createBitmap(int width, int height, Bitmap.Config config)：创建一个宽 width、高 height 的新位图。
- createBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)：从源位图 source 的指定坐标点（给定 x、y）开始，从中“挖取”宽 width、高 height 的一块出来，创建新的 Bitmap 对象。并按 Matrix 指定的规则进行变换。

BitmapFactory 是一个工具类，它用于提供大量的方法，这些方法可用于从不同的数据源来解析、创建 Bitmap 对象。BitmapFactory 包含了如下方法。

- decodeByteArray(byte[] data, int offset, int length)：从指定字节数组的 offset 位置开始，将长度为 length 的字节数据解析成 Bitmap 对象。
- decodeFile(String pathName)：从 pathName 指定的文件中解析、创建 Bitmap 对象。
- decodeFileDescriptor(FileDescriptor fd)：用于从 FileDescriptor 对应的文件中解析、创建 Bitmap 对象。
- decodeResource(Resources res, int id)：用于根据给定的资源 ID 从指定资源中解析、创建 Bitmap 对象。
- decodeStream(InputStream is)：用于从指定输入流中解析、创建 Bitmap 对象。

大部分时候，我们只要把图片放在/res/drawable-mdpi 目录下，就可以在程序中通过该图片对应的资源 ID 来获取封装该图片的 Drawable 对象。但由于手机系统的内存比较小，如果系统不停地去解析、创建 Bitmap 对象，可能由于前面创建 Bitmap 所占用的内存还没有回收，而导致程序运行时引发 OutOfMemory 错误。

Android 为 Bitmap 提供了两个方法来判断它是否已回收，以及强制 Bitmap 回收自己。

- boolean isRecycled()：返回该 Bitmap 对象是否已被回收。
- void recycle()：强制一个 Bitmap 对象立即回收自己。

除此之外，如果 Android 应用需要访问其他存储路径（比如 SD 卡中）里的图片，都需要借助于 BitmapFactory 来解析、创建 Bitmap 对象。

下面开发一个查看/assets/目录下图片的图片查看器，该程序界面十分简单，只包含一个 ImageView 和一个按钮，但用户单击该按钮时程序会自动去搜寻/assets/目录下的下一张图片。此处不再给出界面布局代码，该程序的代码如下。

程序清单：codes\07\7.1\BitmapTest\src\org\crazyit\image\BitmapTest.java

```
public class BitmapTest extends Activity
{
    String[] images = null;
    AssetManager assets = null;
    int currentImg = 0;
    ImageView image;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        image = (ImageView) findViewById(R.id.image);
        try
```

```
        {
            assets = getAssets();
            // 获取/assets/目录下所有文件
            images = assets.list("");
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        // 获取 bn 按钮
        final Button next = (Button) findViewById(R.id.next);
        // 为 bn 按钮绑定事件监听器，该监听器将会查看下一张图片
        next.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View sources)
            {
                // 如果发生数组越界
                if (currentImg >= images.length)
                {
                    currentImg = 0;
                }
                // 找到下一个图片文件
                while (!images[currentImg].endsWith(".png")
                        && !images[currentImg].endsWith(".jpg")
                        && !images[currentImg].endsWith(".gif"))
                {
                    currentImg++;
                    // 如果已发生数组越界
                    if (currentImg >= images.length)
                    {
                        currentImg = 0;
                    }
                }
                InputStream assetFile = null;
                try
                {
                    // 打开指定资源对应的输入流
                    assetFile = assets.open(images[currentImg++]);
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
                BitmapDrawable bitmapDrawable = (BitmapDrawable) image
                        .getDrawable();
                // 如果图片还未回收，先强制回收该图片
                if (bitmapDrawable != null
                    && !bitmapDrawable.getBitmap().isRecycled()) //①
                {
                    bitmapDrawable.getBitmap().recycle();
                }
                // 改变 ImageView 显示的图片
                image.setImageBitmap(BitmapFactory
                    .decodeStream(assetFile)); //②
            }
        });
    }
}
```

上面的程序中①号粗体字代码用于判断当前 ImageView 所显示的图片是否已被回收，如

果该图片还未回收，系统强制回收该图片；程序的②号粗体字代码调用了 BitmapFactory 从指定输入流解析、并创建 Bitmap 对象。

7.2 绘图

除了使用已有的图片之外，Android 应用常常需要在运行时动态地生成图片，比如一个手机游戏，游戏界面看上去丰富多彩，而且可以随着用户动作而动态改变，这就需要借助于 Android 的绘图支持了。

7.2.1 Android 绘图基础：Canvas、Paint 等

有过 Swing 编程经验的读者都知道，在 Swing 中绘图的思路需要开发一个自定义类，该自定义类继承 JPanel，并重写 JPanel 的 paint(Graphics g)方法即可。Android 的绘图与此类似，Android 的绘图应该继承 View 组件，并重写它的 onDraw(Canvas canvas)方法即可。

重写 onDraw(Canvas canvas)方法时涉及一个绘图 API：Canvas，Canvas 代表了“依附”于指定 View 的画布，它提供了如表 7.1 所示的方法绘制各种图形。

表 7.1　Canvas 的绘制方法

方法签名	简要说明
drawArc(RectF oval, float startAngle, float sweepAngle, boolean useCenter, Paint paint)	绘制弧
drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)	在指定点绘制从源位图中“挖取”的一块
drawBitmap(Bitmap bitmap, float left, float top, Paint paint)	在指定点绘制位图
drawCircle(float cx, float cy, float radius, Paint paint)	在指定点绘制一个圆形
drawLine(float startX, float startY, float stopX, float stopY, Paint paint)	绘制一条线
drawLines(float[] pts, int offset, int count, Paint paint)	绘制多条线
drawOval(RectF oval, Paint paint)	绘制椭圆
drawPath(Path path, Paint paint)	沿着指定 Path 绘制任意形状
drawPoint(float x, float y, Paint paint)	绘制一个点
drawPoints(float[] pts, int offset, int count, Paint paint)	绘制多个点
drawRect(float left, float top, float right, float bottom, Paint paint)	绘制矩形
drawRoundRect(RectF rect, float rx, float ry, Paint paint)	绘制圆角矩形
drawText(String text, int start, int end, Paint paint)	绘制字符串
drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint)	沿着路径绘制字符串
clipRect(float left, float top, float right, float bottom)	剪切一个矩形区域
clipRegion(Region region)	剪切指定区域

除了表 7.1 所定义的各种方法之外，Canvas 还提供了如下方法进行坐标变换。

- **rotate(float degrees, float px, float py)**：对 Canvas 执行旋转变换。
- **scale(float sx, float sy, float px, float py)**：对 Canvas 执行缩放变换。
- **skew(float sx, float sy)**：对 Canvas 执行倾斜变换。
- **translate(float dx, float dy)**：移动 Canvas。向右移动 dx 距离（dx 为负数即向左移动）；向下移动 dy 距离（dy 为负数即向上移动）。

Canvas 提供的上面的方法还涉及一个 API：Paint，Paint 代表了 Canvas 上的画笔，因此

Paint 类主要用于设置绘制风格，包括画笔颜色、画笔笔触粗细、填充风格等。Paint 提供了如表 7.2 所示的方法。

表 7.2 Paint 的常用方法

方法签名	简要说明
setARGB(int a, int r, int g, int b)/setColor(int color)	设置颜色
setAlpha(int a)	设置透明度
setAntiAlias(boolean aa)	设置是否抗锯齿
setColor(int color)	设置颜色
setPathEffect(PathEffect effect)	设置绘制路径时的路径效果
setShader(Shader shader)	设置画笔的填充效果
setShadowLayer(float radius, float dx, float dy, int color)	设置阴影
setStrokeWidth(float width)	设置画笔的笔触宽度
setStrokeJoin(Paint.Join join)	设置画笔转弯处的连接风格
setStyle(Paint.Style style)	设置 Paint 的填充风格
setTextAlign(Paint.Align align)	设置绘制文本时的文字的对齐方式
setTextSize(float textSize)	设置绘制文本时的文字大小

在 Canvas 提供的绘制方法中还用到了一个 API：Path，Path 代表任意多条直线连接而成的任意图形，当 Canvas 根据 Path 绘制时，它可以绘制出任意的形状。

下面的程序示范了如何在 Android 应用中绘制基本的集合图形，该程序的关键在于一个自定义 View 组件，程序重写该 View 组件的 onDraw(Canvase)方法，接下来在该 Canvas 上绘制了大量几何图形。这个自定义 View 的代码如下。

程序清单：codes\07\7.2\CanvasTest\src\org\crazyit\image\MyView.java

```
public class MyView extends View
{
    public MyView(Context context, AttributeSet set)
    {
        super(context, set);
    }
    @Override
    // 重写该方法，进行绘图
    protected void onDraw(Canvas canvas)
    {
        super.onDraw(canvas);
        // 把整张画布绘制成白色
        canvas.drawColor(Color.WHITE);
        Paint paint = new Paint();
        // 去锯齿
        paint.setAntiAlias(true);
        paint.setColor(Color.BLUE);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(3);
        // 绘制圆形
        canvas.drawCircle(40, 40, 30, paint);
        // 绘制正方形
        canvas.drawRect(10, 80, 70, 140, paint);
        // 绘制矩形
        canvas.drawRect(10, 150, 70, 190, paint);
        RectF rel = new RectF(10, 200, 70, 230);
        // 绘制圆角矩形
```

```
canvas.drawRoundRect(rel, 15, 15, paint);
RectF rel1 = new RectF(10, 240, 70, 270);
// 绘制椭圆
canvas.drawOval(rel1, paint);
// 定义一个 Path 对象，封闭成一个三角形
Path path1 = new Path();
path1.moveTo(10, 340);
path1.lineTo(70, 340);
path1.lineTo(40, 290);
path1.close();
// 根据 Path 进行绘制，绘制三角形
canvas.drawPath(path1, paint);
// 定义一个 Path 对象，封闭成一个五角形
Path path2 = new Path();
path2.moveTo(26, 360);
path2.lineTo(54, 360);
path2.lineTo(70, 392);
path2.lineTo(40, 420);
path2.lineTo(10, 392);
path2.close();
// 根据 Path 进行绘制，绘制五角形
canvas.drawPath(path2, paint);
// ----------设置填充风格后绘制----------
paint.setStyle(Paint.Style.FILL);
paint.setColor(Color.RED);
canvas.drawCircle(120, 40, 30, paint);
//绘制正方形
canvas.drawRect(90, 80, 150, 140, paint);
//绘制矩形
canvas.drawRect(90, 150, 150, 190, paint);
RectF re2 = new RectF(90, 200, 150, 230);
//绘制圆角矩形
canvas.drawRoundRect(re2, 15, 15, paint);
RectF re21 = new RectF(90, 240, 150, 270);
// 绘制椭圆
canvas.drawOval(re21, paint);
Path path3 = new Path();
path3.moveTo(90, 340);
path3.lineTo(150, 340);
path3.lineTo(120, 290);
path3.close();
//绘制三角形
canvas.drawPath(path3, paint);
Path path4 = new Path();
path4.moveTo(106, 360);
path4.lineTo(134, 360);
path4.lineTo(150, 392);
path4.lineTo(120, 420);
path4.lineTo(90, 392);
path4.close();
//绘制五角形
canvas.drawPath(path4, paint);
// ----------设置渐变器后绘制----------
// 为 Paint 设置渐变器
Shader mShader = new LinearGradient(0, 0, 40, 60
      , new int[] {Color.RED, Color.GREEN, Color.BLUE, Color.YELLOW }
      , null , Shader.TileMode.REPEAT);
paint.setShader(mShader);
//设置阴影
paint.setShadowLayer(45 , 10 , 10 , Color.GRAY);
// 绘制圆形
```

```
            canvas.drawCircle(200, 40, 30, paint);
            // 绘制正方形
            canvas.drawRect(170, 80, 230, 140, paint);
            // 绘制矩形
            canvas.drawRect(170, 150, 230, 190, paint);
            RectF re3 = new RectF(170, 200, 230, 230);
            // 绘制圆角矩形
            canvas.drawRoundRect(re3, 15, 15, paint);
            RectF re31 = new RectF(170, 240, 230, 270);
            // 绘制椭圆
            canvas.drawOval(re31, paint);
            Path path5 = new Path();
            path5.moveTo(170, 340);
            path5.lineTo(230, 340);
            path5.lineTo(200, 290);
            path5.close();
            // 根据 Path 进行绘制，绘制三角形
            canvas.drawPath(path5, paint);
            Path path6 = new Path();
            path6.moveTo(186, 360);
            path6.lineTo(214, 360);
            path6.lineTo(230, 392);
            path6.lineTo(200, 420);
            path6.lineTo(170, 392);
            path6.close();
            // 根据 Path 进行绘制，绘制五角形
            canvas.drawPath(path6, paint);
            // ----------设置字符大小后绘制----------
            paint.setTextSize(24);
            paint.setShader(null);
            // 绘制 7 个字符串
            canvas.drawText(getResources().getString(R.string.circle), 240, 50,
                paint);
            canvas.drawText(getResources().getString(R.string.square), 240, 120,
                paint);
            canvas.drawText(getResources().getString(R.string.rect), 240, 175,
                paint);
            canvas.drawText(getResources().getString(R.string.round_rect), 230,
                220, paint);
            canvas.drawText(getResources().getString(R.string.oval), 240,
                260, paint);
            canvas.drawText(getResources().getString(R.string.triangle), 240, 325,
                paint);
            canvas.drawText(getResources().getString(R.string.pentagon), 240, 390,
                paint);
        }
    }
```

上面的程序中大量调用了 Canvas 的方法来绘制几何图形，而且程序的粗体字代码还为 Paint 画笔设置了使用渐变、阴影，因此接下来绘制的几何图形将采用渐变填充，而且具有阴影。使用一个 Activity 来显示上面的 MyView 类，运行程序将看到如图 7.1 所示的效果。

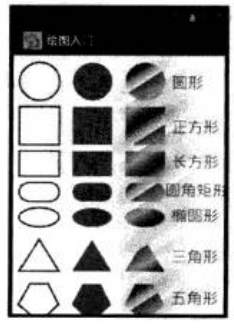


图 7.1 Android 绘图入门

Android 的 Canvas 不仅可以绘制这种简单的几何图形，还可以直接将一个 Bitmap 绘制到画布上，这样就给了开发者巨大的灵活性，只要前期美工把应用程序所需的图片制作出来，后期开发时只要把这些图片绘制到 Canvas 上即可。

7.2.2 Path 类

从上面的程序可以看出，Android 提供的 Path 是一个非常有用的类，它可以预先在 View 上将 N 个点连成一条“路径”，然后调用 Canvas 的 drawPath(path, paint)即可沿着路径绘制图形。实际上 Android 还为路径绘制提供了 PathEffect 来定义绘制效果，PathEffect 包含了如下子类（每个子类代表一种绘制效果）：

- ComposePathEffect
- CornerPathEffect
- DashPathEffect
- DiscretePathEffect
- PathDashPathEffect
- SumPathEffect

这些绘制效果使用语言来表述总显得有点空洞，下面通过一个程序来让读者理解这些绘制效果。接下来的程序绘制 7 条路径，分别示范了不使用效果和使用上面 6 种效果的示意。

程序清单：codes\07\7.2\PathTest\src\org\crazyit\image\PathTest.java

```
public class PathTest extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this));
    }
    class MyView extends View
    {
        float phase;
        PathEffect[] effects = new PathEffect[7];
        int[] colors;
        private Paint paint;
        Path path;
        public MyView(Context context)
        {
            super(context);
            paint = new Paint();
            paint.setStyle(Paint.Style.STROKE);
            paint.setStrokeWidth(4);
            // 创建并初始化 Path
            path = new Path();
            path.moveTo(0, 0);
            for (int i = 1; i <= 15; i++)
            {
                // 生成 15 个点，随机生成它们的 Y 坐标，并将它们连成一条 Path
                path.lineTo(i * 20, (float) Math.random() * 60);
            }
            // 初始化 7 个颜色
            colors = new int[] { Color.BLACK, Color.BLUE, Color.CYAN,
                Color.GREEN, Color.MAGENTA, Color.RED, Color.YELLOW };
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            // 将背景填充成白色
            canvas.drawColor(Color.WHITE);
```

```
                // ----------下面开始初始化 7 种路径效果----------
                // 不使用路径效果
                effects[0] = null;
                // 使用 CornerPathEffect 路径效果
                effects[1] = new CornerPathEffect(10);
                // 初始化 DiscretePathEffect
                effects[2] = new DiscretePathEffect(3.0f, 5.0f);
                // 初始化 DashPathEffect
                effects[3] = new DashPathEffect(new float[] { 20, 10, 5, 10 },
                        phase);
                // 初始化 PathDashPathEffect
                Path p = new Path();
                p.addRect(0, 0, 8, 8, Path.Direction.CCW);
                effects[4] = new PathDashPathEffect(p, 12, phase,
                        PathDashPathEffect.Style.ROTATE);
                // 初始化 PathDashPathEffect
                effects[5] = new ComposePathEffect(effects[2], effects[4]);
                effects[6] = new SumPathEffect(effects[4], effects[3]);
                // 将画布移动到(8、8)处开始绘制
                canvas.translate(8, 8);
                // 依次使用 7 种不同路径效果、7 种不同的颜色来绘制路径
                for (int i = 0; i < effects.length; i++)
                {
                    paint.setPathEffect(effects[i]);
                    paint.setColor(colors[i]);
                    canvas.drawPath(path, paint);
                    canvas.translate(0, 60);
                }
                // 改变 phase 值，形成动画效果
                phase += 1;
                invalidate();
            }
        }
    }
```

正如上面的程序中可以看到的，当定义 DashPathEffect、PathDashPath Effect 时可指定一个 phase 参数，该参数用于指定路径效果的相位，当该 phase 参数改变时，绘制效果也略有变化。上面的程序不停地改变 phase 参数，并不停地重绘该 View 组件，这将看到产生动画效果，如图 7.2 所示。

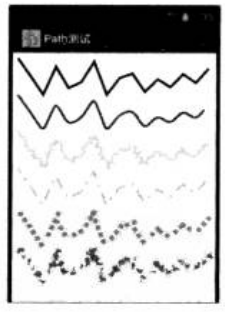


图 7.2 路径效果示例

除此之外，Android 的 Canvas 还提供了一个 drawTextOnPath (String text, Path path, float hOffset, float vOffset, Paint paint)方法，该方法可以沿着 Path 绘制文本。其中 hOffset 参数指定水平偏移、vOffset 参数指定垂直偏移。例如如下程序。

程序清单：codes\07\7.2\PathText\src\org\crazyit\image\PathText.java

```
public class PathText extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new TextView(this));
    }
    class TextView extends View
    {
        final String DRAW_STR = "疯狂 Java 讲义";
```

```
        Path[] paths = new Path[3];
        Paint paint;
        public TextView(Context context)
        {
            super(context);
            paths[0] = new Path();
            paths[0].moveTo(0, 0);
            for (int i = 1; i <= 7; i++)
            {
                // 生成 7 个点，随机生成它们的 Y 坐标，并将它们连成一条 Path
                paths[0].lineTo(i * 30, (float) Math.random() * 30);
            }
            paths[1] = new Path();
            RectF rectF = new RectF(0, 0, 200, 120);
            paths[1].addOval(rectF, Path.Direction.CCW);
            paths[2] = new Path();
            paths[2].addArc(rectF, 60, 180);
            // 初始化画笔
            paint = new Paint();
            paint.setAntiAlias(true);
            paint.setColor(Color.CYAN);
            paint.setStrokeWidth(1);
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            canvas.drawColor(Color.WHITE);
            canvas.translate(40, 40);
            // 设置从右边开始绘制（右对齐）
            paint.setTextAlign(Paint.Align.RIGHT);
            paint.setTextSize(20);
            // 绘制路径
            paint.setStyle(Paint.Style.STROKE);
            canvas.drawPath(paths[0], paint);
            // 沿着路径绘制一段文本
            paint.setStyle(Paint.Style.FILL);
            canvas.drawTextOnPath(DRAW_STR, paths[0], -8, 20, paint);
            // 对 Canvas 进行坐标变换：画布下移 120
            canvas.translate(0, 60);
            // 绘制路径
            paint.setStyle(Paint.Style.STROKE);
            canvas.drawPath(paths[1], paint);
            // 沿着路径绘制一段文本
            paint.setStyle(Paint.Style.FILL);
            canvas.drawTextOnPath(DRAW_STR, paths[1], -20, 20, paint);
            // 对 Canvas 进行坐标变换： 画布下移 120
            canvas.translate(0, 120);
            // 绘制路径
            paint.setStyle(Paint.Style.STROKE);
            canvas.drawPath(paths[2], paint);
            // 沿着路径绘制一段文本
            paint.setStyle(Paint.Style.FILL);
            canvas.drawTextOnPath(DRAW_STR, paths[2], -10, 20, paint);
        }
    }
}
```

上面的程序三次调用了 drawTextOnPath 在 View 组件上绘制文本，此时绘制的文本并不是简单地水平排列，而是沿着指定路径绘制的。运行上面的程序将看到如图 7.3 所示的结果。

7.2.3 绘制游戏动画



图 7.3 沿着路径绘制文本

掌握了 Canvas 绘图之后，如果需要实现游戏动画也是非常简单的。动画其实就是不断地重复调用 View 组件的 onDraw(Canvas canvas)方法，如果每次在 View 组件上绘制的图形并不相同，就成了习惯上所说的动画。

为了让 View 组件上绘制的图形发生改变（无非是位置、大小、角度等发生改变），这就需要程序采用变量来“记住”这些状态数据——如果需要游戏动画随用户操作而改变，就为用户动作编写事件监听器，在监听器中修改这些数据；如果需要游戏动画“自动”改变，那就是随时间的流失而改变，就需要使用定时器（Timer），让 Timer 控制这些状态数据定期修改。

不管使用哪种方式，每次 View 组件上的图形状态数据发生了改变，都应该通知 View 组件重写调用 onDraw(Canvas canvas)方法重绘该组件。通知 View 重绘可调用 invalidate（在 UI 线程中）或 postInvalidate（在非 UI 线程中）。

实例：采用双缓冲实现画图板

本实例要实现一个画图板，当用户在触摸屏上移动时，即可在屏幕上绘制任意的图形。实现手绘功能其实是一种假象：表面上看起来可以随用户在触摸屏上自由地画曲线，实际上依然利用的是 Canvas 的 drawLine 方法画直线，每条直线都是从上一次拖动事件发生点画到本次拖动事件发生点。当用户在触摸屏上移动时，两次拖动事件发生点的距离很小，多条极短的直线连接起来，肉眼看起来就是直线了。借助于 Android 提供的 Path 类，可以非常方便地实现这种效果。

需要指出的是，如果程序每次都只是从上次拖动事件的发生点绘一条直线到本次拖动事件的发生点，那么用户前面绘制的就会丢失。为了保留用户之前绘制的内容，程序要借助于“双缓冲”技术。

所谓双缓冲技术其实很简单：当程序需要在指定 View 上进行绘制时，程序并不直接绘制到该 View 组件上，而是先绘制到一个内存中的 Bitmap 图片（这就是缓冲）上，等到内存中的 Bitmap 绘制好之后，再一次性地将 Bitmap 绘制到 View 组件上。

该程序需要一个自定义 View，该 View 的代码如下。

程序清单：codes\07\7.2\HandDraw\src\org\crazyit\image\DrawView.java

```
public class DrawView extends View
{
    float preX;
    float preY;
    private Path path;
    public Paint paint = null;
    final int VIEW_WIDTH = 320;
    final int VIEW_HEIGHT = 480;
    // 定义一个内存中的图片，该图片将作为缓冲区
    Bitmap cacheBitmap = null;
    // 定义 cacheBitmap 上的 Canvas 对象
    Canvas cacheCanvas = null;
    public DrawView(Context context, AttributeSet set)
    {
        super(context, set);
```

```
        // 创建一个与该 View 相同大小的缓存区
        cacheBitmap = Bitmap.createBitmap(VIEW_WIDTH, VIEW_HEIGHT,
                Config.ARGB_8888);
        cacheCanvas = new Canvas();
        path = new Path();
        // 设置 cacheCanvas 将会绘制到内存中的 cacheBitmap 上
        cacheCanvas.setBitmap(cacheBitmap);
        // 设置画笔的颜色
        paint = new Paint(Paint.DITHER_FLAG);
        paint.setColor(Color.RED);
        // 设置画笔风格
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeWidth(1);
        // 反锯齿
        paint.setAntiAlias(true);
        paint.setDither(true);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 获取拖动事件的发生位置
        float x = event.getX();
        float y = event.getY();
        switch (event.getAction())
        {
            case MotionEvent.ACTION_DOWN:
                path.moveTo(x, y);
                preX = x;
                preY = y;
                break;
            case MotionEvent.ACTION_MOVE:
                path.quadTo(preX, preY, x, y);
                preX = x;
                preY = y;
                break;
            case MotionEvent.ACTION_UP:
                cacheCanvas.drawPath(path, paint); // ①
                path.reset();
                break;
        }
        invalidate();
        // 返回 true 表明处理方法已经处理该事件
        return true;
    }
    @Override
    public void onDraw(Canvas canvas)
    {
        Paint bmpPaint = new Paint();
        // 将 cacheBitmap 绘制到该 View 组件上
        canvas.drawBitmap(cacheBitmap, 0, 0, bmpPaint); // ②
        // 沿着 path 绘制
        canvas.drawPath(path, paint);
    }
}
```

上面的程序中粗体字代码为触摸屏的拖动事件提供了响应——只是简单地修改了 preX、preY 两个属性，并通知该组件重绘。

在这个自定义 View 组件中，程序重写了该 View 的 onDraw(Canvas canvas)方法，注意该方法中的①号代码，这行代码并不是调用该 View 的 Canvas 进行绘制，而是调用了缓存 Bitmap

的 Canvas 进行绘图，这表明是向缓冲绘图。程序的 ②号粗体字代码将缓冲中的 BitMap 对象绘制到 View 组件上——这就是所谓的“双缓冲”技术。

提供了上面的 DrawView 之后，接下来把该组件添加到界面布局中。界面布局代码很简单，此处不再给出。

本程序还提供了菜单来设置画笔的颜色和笔触大小，该程序的菜单资源文件如下。

程序清单：codes\07\7.2\HandDraw\res\menu\my_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:title="@string/color">
        <menu>
            <!-- 定义一组单选菜单项 -->
            <group android:checkableBehavior="single">
                <!-- 定义多个菜单项 -->
                <item
                    android:id="@+id/red" android:title="@string/color_red"/>
                <item
                    android:id="@+id/green" android:title="@string/color_green"/>
                <item
                    android:id="@+id/blue" android:title="@string/color_blue"/>
            </group>
        </menu>
    </item>
    <item android:title="@string/width">
        <menu>
            <!-- 定义一组菜单项 -->
            <group>
                <!-- 定义三个菜单项 -->
                <item
                    android:id="@+id/width_1" android:title="@string/width_1"/>
                <item
                    android:id="@+id/width_3" android:title="@string/width_3"/>
                <item
                    android:id="@+id/width_5" android:title="@string/width_5"/>
            </group>
        </menu>
    </item>
    <item android:id="@+id/blur" android:title="@string/blur"/>
    <item android:id="@+id/emboss" android:title="@string/emboss"/>
                        </menu>
```

主程序负责加载、显示界面布局，加载、显示上面的菜单资源。除此之外，程序还要为各菜单项编写事件响应，程序代码如下。

程序清单：codes\07\7.2\HandDraw\src\org\crazyit\image\HandDraw.java

```
public class HandDraw extends Activity
{
    EmbossMaskFilter emboss;
    BlurMaskFilter blur;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        emboss = new EmbossMaskFilter(new float[]
            { 1.5f, 1.5f, 1.5f }, 0.6f,    6, 4.2f);
        blur = new BlurMaskFilter(8, BlurMaskFilter.Blur.NORMAL);
    }
```

```
@Override
// 负责创建选项菜单
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflator = new MenuInflater(this);
    // 装载 R.menu.my_menu 对应的菜单，并添加到 menu 中
    inflator.inflate(R.menu.my_menu, menu);
    return super.onCreateOptionsMenu(menu);
}
@Override
// 菜单项被单击后的回调方法
public boolean onOptionsItemSelected(MenuItem mi)
{
    DrawView dv = (DrawView) findViewById(R.id.draw);
    // 判断单击的是哪个菜单项，并有针对性地作出响应
    switch (mi.getItemId())
    {
        case R.id.red:
            dv.paint.setColor(Color.RED);
            mi.setChecked(true);
            break;
        case R.id.green:
            dv.paint.setColor(Color.GREEN);
            mi.setChecked(true);
            break;
        case R.id.blue:
            dv.paint.setColor(Color.BLUE);
            mi.setChecked(true);
            break;
        case R.id.width_1:
            dv.paint.setStrokeWidth(1);
            break;
        case R.id.width_3:
            dv.paint.setStrokeWidth(3);
            break;
        case R.id.width_5:
            dv.paint.setStrokeWidth(5);
            break;
        case R.id.blur:
            dv.paint.setMaskFilter(blur);
                    break;
                case R.id.emboss:
                    dv.paint.setMaskFilter(emboss);
                    break;
            }
            return true;
        }
    }
```

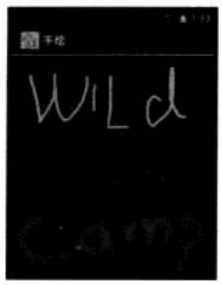


图 7.4　双缓冲实现绘图板

上面的程序代码比较简单，当用户单击不同菜单项之后，程序只要简单地修改 DrawView 组件内的 Paint 对象的颜色和笔触粗细即可。运行上面的程序，将看到如图 7.4 所示的界面。



提示：

阅读过《疯狂 Java 讲义》一书的读者可能对这个程序感到很“眼熟”，实际上这个程序所用的“双缓冲”技术，编程思路都来自《疯狂 Java 讲义》一书中 11.8 节的示例。正如笔者前面提到的，Android 开发并不难——如果读者有扎实的 Java 基础，再加上一定的界面编程经验，学习 Android 开发将非常轻松。

实例：弹球游戏

下面的程序开发了一个简单的弹球游戏，其中小球和球拍分别以圆形区域和矩形区域代替，小球开始以随机速度向下运动，遇到边框或球拍时小球反弹；球拍则由用户控制，当用户按下向左、向右键时，球拍将会向左、向右移动。程序如下。

程序清单：codes\07\7.2\PinBall\src\org\crazyit\image\PinBall.java

```
public class PinBall extends Activity
{
    // 桌面的宽度
    private int tableWidth;
    // 桌面的高度
    private int tableHeight;
    // 球拍的垂直位置
    private int racketY;
    // 下面定义球拍的高度和宽度
    private final int RACKET_HEIGHT = 20;
    private final int RACKET_WIDTH = 70;
    // 小球的大小
    private final int BALL_SIZE = 12;
    // 小球纵向的运行速度
    private int ySpeed = 10;
    Random rand = new Random();
    // 返回一个-0.5~0.5的比率，用于控制小球的运行方向
    private double xyRate = rand.nextDouble() - 0.5;
    // 小球横向的运行速度
    private int xSpeed = (int) (ySpeed * xyRate * 2);
    // ballX和ballY代表小球的坐标
    private int ballX = rand.nextInt(200) + 20;
    private int ballY = rand.nextInt(10) + 20;
    // racketX代表球拍的水平位置
    private int racketX = rand.nextInt(200);
    // 游戏是否结束的旗标
    private boolean isLose = false;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 去掉窗口标题
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        // 全屏显示
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
        // 创建GameView组件
        final GameView gameView = new GameView(this);
        setContentView(gameView);
        // 获取窗口管理器
        WindowManager windowManager = getWindowManager();
        Display display = windowManager.getDefaultDisplay();
        DisplayMetrics metrics = new DisplayMetrics();
        display.getMetrics(metrics);
        // 获得屏幕宽和高
        tableWidth = metrics.widthPixels;
        tableHeight = metrics.heightPixels;
        racketY = tableHeight - 80;
        final Handler handler = new Handler()
        {
            public void handleMessage(Message msg)
```

```
        {
            if (msg.what == 0x123)
            {
                gameView.invalidate();
            }
        }
    };
    gameView.setOnKeyListener(new OnKeyListener() // ②
    {
        @Override
        public boolean onKey(View source, int keyCode, KeyEvent event)
        {
            // 获取由哪个键触发的事件
            switch (event.getKeyCode())
            {
                // 控制挡板左移
                case KeyEvent.KEYCODE_A:
                    if (racketX > 0) racketX -= 10;
                    break;
                // 控制挡板右移
                case KeyEvent.KEYCODE_D:
                    if (racketX < tableWidth - RACKET_WIDTH) racketX += 10;
                    break;
            }
            // 通知 gameView 组件重绘
            gameView.invalidate();
            return true;
        }
    });
    final Timer timer = new Timer();
    timer.schedule(new TimerTask() // ①
        {
            @Override
            public void run()
            {
                // 如果小球碰到左边边框
                if (ballX <= 0 || ballX >= tableWidth - BALL_SIZE)
                {
                    xSpeed = -xSpeed;
                }
                // 如果小球高度超出了球拍位置，且横向不在球拍范围之内，游戏结束
                if (ballY >= racketY - BALL_SIZE
                        && (ballX < racketX || ballX > racketX
                                + RACKET_WIDTH))
                {
                    timer.cancel();
                    // 设置游戏是否结束的旗标为 true
                    isLose = true;
                }
                // 如果小球位于球拍之内，且到达球拍位置，小球反弹
                else if (ballY <= 0
                    || (ballY >= racketY - BALL_SIZE
                        && ballX > racketX && ballX <= racketX
                        + RACKET_WIDTH))
                {
                    ySpeed = -ySpeed;
                }
                // 小球坐标增加
                ballY += ySpeed;
                ballX += xSpeed;
                // 发送消息，通知系统重绘组件
```

```
                        handler.sendEmptyMessage(0x123);
                    }
                }, 0, 100);
        }
        class GameView extends View
        {
            Paint paint = new Paint();
            public GameView(Context context)
            {
                super(context);
                setFocusable(true);
            }
            // 重写 View 的 onDraw 方法，实现绘画
            public void onDraw(Canvas canvas)
            {
                paint.setStyle(Paint.Style.FILL);
                // 设置去锯齿
                paint.setAntiAlias(true);
                // 如果游戏已经结束
                if (isLose)
                {
                    paint.setColor(Color.RED);
                    paint.setTextSize(40);
                    canvas.drawText("游戏已结束", 50, 200, paint);
                }
                // 如果游戏还未结束
                else
                {
                    // 设置颜色，并绘制小球
                    paint.setColor(Color.rgb(240, 240, 80));
                    canvas.drawCircle(ballX, ballY, BALL_SIZE, paint);
                    // 设置颜色，并绘制球拍
                    paint.setColor(Color.rgb(80, 80, 200));
                    canvas.drawRect(racketX, racketY, racketX + RACKET_WIDTH,
                            racketY + RACKET_HEIGHT, paint);
                }
            }
        }
    }
```

上面的程序提供了一个 GameView，这个 GameView 很简单，它只是根据程序中小球坐标、球拍坐标来绘制小球和球拍。

除此之外，这个所谓的“游戏”本质上就是一个动画程序，该程序中可以“动”的内容很少：只有一个小球和一个球拍。其中小球的坐标由定时器定时修改（如程序中①号代码所示）；球拍坐标则由键盘动作来修改（如程序中②号代码所示）。运行上面的程序，将可以看到如图 7.5 所示的游戏界面。

虽然图 7.5 所示的弹球游戏还略嫌粗糙，但只要开发者找到一些“美丽”的图片，替换程序中的小球、球拍，再考虑为界面上方增加一些“障碍物”来增加游戏乐趣，再为小球碰撞边界、碰撞球拍时增加音效，这个游戏将会变得“生动”起来。这个游戏的 AWT 版本同样来自于《疯狂 Java 讲义》，疯狂 Java 联盟（crazyit.org）站点上有大量已经完成的弹球游戏，读者也可利用那些游戏中的图片资源来完善这个游戏。

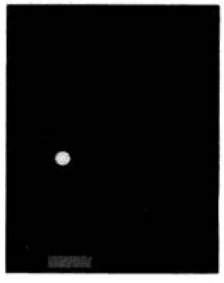


图 7.5 弹球游戏界面

7.3. 图形特效处理

Android 除了前面介绍的这些图形支持之外，还提供了一些额外的更高级的图形特效支持，这些图形特效支持可以让开发者开发出更“绚丽”的 UI 界面。

7.3.1 使用 Matrix 控制变换

Matrix 是 Android 提供的一个矩阵工具类，它本身并不能对图像或组件进行变换，但它可与其他 API 结合来控制图形、组件的变换。

使用 Matrix 控制图像或组件变换的步骤如下。

① 获取 Matrix 对象，该 Matrix 对象既可新创建，也可直接获取其他对象内封装的 Matrix（例如 Transformation 对象内部就封装了 Matrix）。

② 调用 Matrix 的方法进行平移、旋转、缩放、倾斜等。

③ 将程序对 Matrix 所做的变换应用到指定图像或组件。



提示：

从这里的介绍可以看出，Matrix 不仅可用于控制图形的平移、旋转、缩放、倾斜变换，也可控制 View 组件进行平移、旋转和缩放等。

Matrix 提供了如下方法来控制平移、旋转和缩放：

- setTranslate(float dx, float dy)：控制 Matrix 进行平移。
- setSkew(float kx, float ky, float px, float py)：控制 Matrix 以 px、py 为轴心进行倾斜。kx、ky 为 X、Y 方向上的倾斜距离。
- setSkew(float kx, float ky)：控制 Matrix 进行倾斜。kx、ky 为 X、Y 方向上的倾斜距离。
- setRotate(float degrees)：控制 Matrix 进行旋转，degrees 控制旋转的角度。
- setRotate(float degrees, float px, float py)：设置以 px、py 为轴心进行旋转，degrees 控制旋转的角度。
- setScale(float sx, float sy)：设置 Matrix 进行缩放，sx、sy 控制 X、Y 方向上的缩放比例。
- setScale(float sx, float sy, float px, float py)：设置 Matrix 以 px、py 为轴心进行缩放，sx、sy 控制 X、Y 方向上的缩放比例。

一旦对 Matrix 进行了变换，接下来就可应用该 Matrix 对图形进行控制了。例如 Canvas 就提供了一个 drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint)方法，调用该方法就可以在绘制 bitmap 时应用 Matrix 上的变换。

如下程序开发了一个自定义 View，该自定义 View 可以检测到用户的键盘事件，当用户单击手机的方向键时，该自定义 View 会用 Matrix 对绘制的图形进行旋转、倾斜变换。

程序清单：codes\07\7.3\MatrixTest\src\org\crazyit\image\MyView.java

```
public class MyView extends View
{
    // 初始的图片资源
    private Bitmap bitmap;
    // Matrix 实例
```

```
private Matrix matrix = new Matrix();
// 设置倾斜度
private float sx = 0.0f;
// 位图宽和高
private int width, height;
// 缩放比例
private float scale = 1.0f;
// 判断缩放还是旋转
private boolean isScale = false;
public MyView(Context context , AttributeSet set)
{
      super(context , set);
      // 获得位图
      bitmap = ((BitmapDrawable) context.getResources().getDrawable(
            R.drawable.a)).getBitmap();
      // 获得位图宽
      width = bitmap.getWidth();
      // 获得位图高
      height = bitmap.getHeight();
      // 使当前视图获得焦点
      this.setFocusable(true);
}
@Override
protected void onDraw(Canvas canvas)
{
      super.onDraw(canvas);
      // 重置 Matrix
      matrix.reset();
      if (!isScale)
      {
            // 旋转 Matrix
            matrix.setSkew(sx, 0);
      }
      else
      {
            // 缩放 Matrix
            matrix.setScale(scale, scale);
      }
      // 根据原始位图和 Matrix 创建新图片
      Bitmap bitmap2 = Bitmap.createBitmap(bitmap, 0, 0, width, height,
            matrix, true);
      // 绘制新位图
      canvas.drawBitmap(bitmap2, matrix, null);
}
@Override
public boolean onKeyDown(int keyCode, KeyEvent event)
{
      switch(keyCode)
      {
            // 向左倾斜
            case KeyEvent.KEYCODE_A:
                  isScale = false;
                  sx += 0.1;
                  postInvalidate();
                  break;
            // 向右倾斜
            case KeyEvent.KEYCODE_D:
                  isScale = false;
                  sx -= 0.1;
                  postInvalidate();
                  break;
```

```
                // 放大
                case KeyEvent.KEYCODE_W:
                    isScale = true;
                    if (scale < 2.0)
                        scale += 0.1;
                    postInvalidate();
                    break;
                // 缩小
                case KeyEvent.KEYCODE_S:
                    isScale = true;
                    if (scale > 0.5)
                        scale -= 0.1;
                    postInvalidate();
                    break;
            }
            return super.onKeyDown(keyCode, event);
        }
    }
```



图 7.6　使用 Matrix 控制倾斜

上面的程序中粗体字代码就是通过 Matrix 控制图片倾斜、缩放的关键代码，当用户单击手机的方向键时，事件处理器负责修改程序中 sx（控制水平倾斜度）和 scale（控制缩放比）两个参数。

把上面的自定义 View 放在 Activity 中显示出来，运行该程序将看到如图 7.6 所示的界面。

实例：移动游戏背景

• 借助于 Bitmap 的 createBitmap 方法可以“挖取”源位图的其中一块，这样可以在程序中通过定时器控制不断地“挖取”源位图不同位置的块，从而给用户看到背景移动“假象”。

假设要开发经典“雷电”飞机游戏，为了给用户造成飞机不断飞行的错觉，可以通过这种方式来控制背景图片不断下移，用户就会感觉飞机在不断地向上飞行。

例如如下程序实现了背景图片不断“下移”的效果，用户会感觉飞机在不断地向上飞行。

程序清单：codes\07\7.3\MoveBack\src\org\crazyit\image\MoveBack.java

```
public class MoveBack extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this));
    }
    class MyView extends View
    {
        // 记录背景位图的实际高度
        final int BACK_HEIGHT = 1700;
        // 背景图片
        private Bitmap back;
        private Bitmap plane;
        // 背景图片的开始位置
        final int WIDTH = 320;
        final int HEIGHT = 440;
        private int startY = BACK_HEIGHT - HEIGHT;
        public MyView(Context context)
        {
            super(context);
```

```
            back = BitmapFactory.decodeResource(context.getResources(),
                R.drawable.back_img);
            plane = BitmapFactory.decodeResource(context.getResources(),
                R.drawable.plane);
            final Handler handler = new Handler()
            {
                public void handleMessage(Message msg)
                {
                    if (msg.what == 0x123)
                    {
                        // 重新开始移动
                        if (startY <= 0)
                        {
                            startY = BACK_HEIGHT - HEIGHT;
                        }
                        else
                        {
                            startY -= 3;
                        }
                    }
                    invalidate();
                }
            };
            new Timer().schedule(new TimerTask()
            {
                @Override
                public void run()
                {
                    handler.sendEmptyMessage(0x123);
                }
            }, 0, 100);
        }
        @Override
        public void onDraw(Canvas canvas)
        {
            // 根据原始位图和 Matrix 创建新图片
            Bitmap bitmap2 = Bitmap
                .createBitmap(back, 0, startY, WIDTH, HEIGHT); // ①
            // 绘制新位图
            canvas.drawBitmap(bitmap2, 0, 0, null);
            // 绘制飞机
            canvas.drawBitmap(plane, 160, 380, null);
        }
    }
}
```

上面的程序中①号粗体字代码根据 startY 控制“挖取”back 位图中不同位置的块，并将它绘制在当前 View 上。程序的第一段粗体字代码则通过定时器来控制 startY 不断地改变，这样即可看到 back 图片不断下移的效果。运行该程序，读者即可看到飞机向上飞行的“错觉”。



提示：

该程序并未为用户按键事件提供监听器，实际上读者完全可以为该 View 的按键事件绑定监听器，这样用户即可控制飞机飞行。这已经是雷电游戏的雏型了。如果读者想要自己更好地完善这个雷电游戏，可以从如下三个方面进行改进：

（1）为该 View 添加一个数组来控制随机出现的敌机的坐标，当然也要通过定时器来控制敌机坐标的改变。

（2）为该 View 添加一个数组来控制飞机所发出的子弹坐标，当然也要通过定时器来控制敌机坐标的改变。

（3）定期检查子弹是否与敌机“碰撞”，检查敌机是否与自己的飞机“碰撞”，如果发生碰撞，程序在“碰撞”点播放“爆炸”动画。本章后面马上就会介绍在指定点播放“爆炸”动画的示例。

7.3.2 使用 drawBitmapMesh 扭曲图像

Canvas 提供了一个 drawBitmapMesh(Bitmap bitmap, int meshWidth, int meshHeight, float[] verts, int vertOffset, int[] colors, int colorOffset, Paint paint)方法，该方法可以对 bitmap 进行扭曲。这个方法非常灵活，如果用好这个方法，开发者可以在 Android 应用上开发出“水波荡漾”、“风吹旗帜”等各种扭曲效果。

drawBitmapMesh 方法关键参数的说明如下。

- bitmap：指定需要扭曲的源位图。
- meshWidth：该参数控制在横向上把该源位图划分成多少格。
- meshHeight：该参数控制在纵向上把该源位图划分成多少格。
- verts：该参数是一个长度为(meshWidth+1)*(meshHeight+1)*2 的数组，它记录了扭曲后的位图各“顶点”（图 7.7 所示网格线的交点）位置。虽然它是个一维数组，实际上它记录的数据是形如(x0,y0)、(x1,y1)、(x2,y2).....(x*N*,y*N*)格式的数据，这些数组元素控制对 bitmap 位图的扭曲效果。
- vertOffset：控制 verts 数组中从第几个数组元素开始才对 bitmap 进行扭曲（忽略 verOffset 之前数据的扭曲效果）。

drawBitmapMesh 方法对源位图扭曲时最关键的参数是 meshWidth、meshHeight、verts，这三个参数对扭曲的控制如图 7.7 所示。

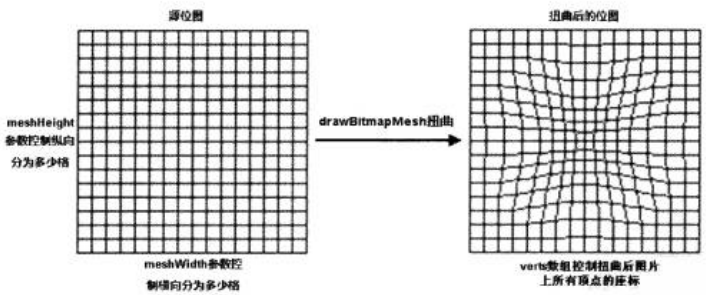


图 7.7　drawBitmapMesh 扭曲示意

从图 7.7 可以看出，当程序希望调用 drawBitmapMesh 方法对位图进行扭曲时，关键是计算 verts 数组的值——该数组的值记录了扭曲后的位图上各“顶点”（图 7.7 所示网格线的交点）的坐标。



提示：

初学者往往容易对 drawBitmapMesh 方法感到不好理解，如果读者有 Photoshop 图形处理的经验，应该对图 7.7 所示的扭曲效果很熟悉——实际上该方法就可模拟 Photoshop 的扭曲“滤镜”。Android 应用面向的终端用户都是普通人群，应用界面对这些用户的影响非常大，Android 提供的 drawBitmapMesh 方法带给开发者一种非常灵活的控制。在实际开发中，笔者甚至“贪心”地希望 Android 提供更多方法，这些方法可以更好地模拟 Photoshop 的各种“滤镜”。

实例：可揉动的图片

下面的实例程序将会通过 drawBitmapMesh 方法来控制图片的扭曲，当用户“触摸”图片的指定点时，该图片会在这个点被用户“按”下去——就像这张图片铺在“极软的床上”一样。

为了实现这个效果，程序要在用户触摸图片的指定点时，动态地改变 verts 数组里每个元素的位置（控制扭曲后每个顶点的坐标）——这种改变也简单：程序计算图片上每个顶点与触摸点的距离，顶点与触摸点的距离越小，该顶点向触摸点移动的距离越大。

下面是该程序的代码。

程序清单：codes\07\7.3\MeshTest\src\org\crazyit\image\WarpTest.java

```
public class WarpTest extends Activity
{
    private Bitmap bitmap;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(new MyView(this, R.drawable.jinta));
    }
    private class MyView extends View
    {
        // 定义两个常量，这两个常量指定该图片横向、纵向上都被划分为 20 格
        private final int WIDTH = 20;
        private final int HEIGHT = 20;
        // 记录该图片上包含 441 个顶点
        private final int COUNT = (WIDTH + 1) * (HEIGHT + 1);
        // 定义一个数组，保存 Bitmap 上的 21 * 21 个点的坐标
        private final float[] verts = new float[COUNT * 2];
        // 定义一个数组，记录 Bitmap 上的 21 * 21 个点经过扭曲后的坐标
        // 对图片进行扭曲的关键就是修改该数组里元素的值
        private final float[] orig = new float[COUNT * 2];
        public MyView(Context context, int drawableId)
        {
            super(context);
            setFocusable(true);
            // 根据指定资源加载图片
            bitmap = BitmapFactory.decodeResource(getResources()
                , drawableId);
            // 获取图片宽度、高度
            float bitmapWidth = bitmap.getWidth();
            float bitmapHeight = bitmap.getHeight();
            int index = 0;
            for (int y = 0; y <= HEIGHT; y++)
            {
                float fy = bitmapHeight * y / HEIGHT;
                for (int x = 0; x <= WIDTH; x++)
                {
                    float fx = bitmapWidth * x / WIDTH;
                    // 初始化 orig、verts 数组。 初始化后，orig、verts
                    // 两个数组均匀地保存了 21 * 21 个点的 x,y 坐标
                    orig[index * 2 + 0] = verts[index * 2 + 0] = fx;
                    orig[index * 2 + 1] = verts[index * 2 + 1] = fy;
                    index += 1;
                }
            }
```

```
        // 设置背景色
        setBackgroundColor(Color.WHITE);
    }
    @Override
    protected void onDraw(Canvas canvas)
    {
        //对 bitmap 按 verts 数组进行扭曲
        //从第一个点（由第 5 个参数 0 控制）开始扭曲
        canvas.drawBitmapMesh(bitmap, WIDTH, HEIGHT, verts
            , 0, null, 0,null);
    }
    // 工具方法，用于根据触摸事件的位置计算 verts 数组里各元素的值
    private void warp(float cx, float cy)
    {
        for (int i = 0; i < COUNT * 2; i += 2)
        {
            float dx = cx - orig[i + 0];
            float dy = cy - orig[i + 1];
            float dd = dx * dx + dy * dy;
            // 计算每个坐标点与当前点（cx、cy）之间的距离
            float d = (float) Math.sqrt(dd);
            // 计算扭曲度，距离当前点（cx、cy）越远，扭曲度越小
            float pull = 80000 / ((float) (dd * d));
            // 对 verts 数组（保存 bitmap 上 21 * 21 个点经过扭曲后的坐标）重新赋值
            if (pull >= 1)
            {
                verts[i + 0] = cx;
                verts[i + 1] = cy;
            }
            else
            {
                // 控制各顶点向触摸事件发生点偏移
                verts[i + 0] = orig[i + 0] + dx * pull;
                verts[i + 1] = orig[i + 1] + dy * pull;
            }
        }
        // 通知 View 组件重绘
        invalidate();
    }
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        // 调用 warp 方法根据触摸屏事件的坐标点来扭曲 verts 数组
        warp(event.getX(), event.getY());
        return true;
    }
}
}
```



图 7.8　扭曲图像

上面的程序中粗体字代码是关键，该方法将会根据触摸点的位置（由 cx、cy 坐标控制）动态修改 verts 数组里所有数组元素的值，这样就控制了 drawBitmapMesh 方法的扭曲效果。

运行该程序并触碰该图片的任意位置，将可以看到如图 7.8 所示的效果。

7.3.3　使用 Shader 填充图形

前面介绍 Paint 时提到该 Shader 包含了一个 setShader(Shader s)方法，

该方法控制“画笔”的渲染效果：Android 不仅可以使用颜色来填充图形（包括前面介绍的矩形、椭圆、圆形等各种几何图形），也可以使用 Shader 对象指定的渲染效果来填充图形。

Shader 本身是一个抽象类，它提供了如下实现类。

- BitmapShader：使用位图平铺的渲染效果。
- LinearGradient：使用线性渐变来填充图形。
- RadialGradient：使用圆形渐变来填充图形。
- SweepGradient：使用角度渐变来填充图形。
- ComposeShader：使用组合渲染效果来填充图形。

如果使用文字来描述这些渲染效果，不仅显得十分啰嗦而且极难讲清楚。但如果读者有“玩”Flash 的经验，应该对位图平铺、线性渐变、圆形渐变、角度渐变等名词十分熟悉，那么此处就无须笔者多费笔墨来描述这些渲染效果了，因为它们与 Flash 提供位图平铺、线性渐变、圆形渐变、角度渐变完全一样。如果读者没有 Flash 经验，也无须担心，运行下面的程序即可明白各种 Shader 对象的渲染效果。

下面的程序中包含了 5 个按钮，当用户单击不同按钮时系统将会设置 Paint 使用不同的 Shader，这样读者即可看到不同 Shader 的效果。

程序清单：codes\07\7.3\ShaderTest\src\org\crazyit\image\ShaderTest.java

```
public class ShaderTest extends Activity
    implements OnClickListener
{
    // 声明位图渲染对象
    private Shader[] shaders = new Shader[5];
    // 声明颜色数组
    private int[] colors;
    MyView myView;
    // 自定义视图类
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myView = (MyView)findViewById(R.id.my_view);
        // 获得 Bitmap 实例
        Bitmap bm = BitmapFactory.decodeResource(getResources()
            , R.drawable.water);
        // 设置渐变的颜色组，也就是按红、绿、蓝的方式渐变
        colors = new int[] { Color.RED, Color.GREEN, Color.BLUE };
        // 实例化 BitmapShader，x 坐标方向重复图形，y 坐标方向镜像图形
        shaders[0] = new BitmapShader(bm, TileMode.REPEAT,
            TileMode.MIRROR);
        // 实例化 LinearGradient
        shaders[1] = new LinearGradient(0, 0, 100, 100
            , colors, null, TileMode.REPEAT);
        // 实例化 RadialGradient
        shaders[2] = new RadialGradient(100, 100, 80, colors, null,
            TileMode.REPEAT);
        // 实例化 SweepGradient
        shaders[3] = new SweepGradient(160, 160, colors, null);
        // 实例化 ComposeShader
        shaders[4] = new ComposeShader(shaders[1], shaders[2],
            PorterDuff.Mode.DARKEN);
        Button bn1 = (Button)findViewById(R.id.bn1);
        Button bn2 = (Button)findViewById(R.id.bn2);
        Button bn3 = (Button)findViewById(R.id.bn3);
```

```
        Button bn4 = (Button)findViewById(R.id.bn4);
        Button bn5 = (Button)findViewById(R.id.bn5);
        bn1.setOnClickListener(this);
        bn2.setOnClickListener(this);
        bn3.setOnClickListener(this);
        bn4.setOnClickListener(this);
        bn5.setOnClickListener(this);
    }
    @Override
    public void onClick(View source)
    {
        switch(source.getId())
        {
            case R.id.bn1:
                myView.paint.setShader(shaders[0]);
                break;
            case R.id.bn2:
                myView.paint.setShader(shaders[1]);
                break;
            case R.id.bn3:
                myView.paint.setShader(shaders[2]);
                break;
            case R.id.bn4:
                myView.paint.setShader(shaders[3]);
                break;
            case R.id.bn5:
                myView.paint.setShader(shaders[4]);
                break;
        }
        // 重绘界面
        myView.invalidate();
    }
}
```

上面的程序中第一段粗体字代码创建多个 Shader 对象，第二段粗体字代码会根据用户按下不同的按钮来修改 MyView 对象 Paint——MyView 对象将会用该 Paint 来绘制一个矩形。随着 MyView 的 Paint 所使用的 Shader 不断改变，MyView 对象所绘制的矩形也随之改变。

运行上面的程序，如果使用 BitmapShader 绘制矩形，将看到如图 7.9 所示的效果。

如果使用 SweepGradient 绘制矩形，将看到如图 7.10 所示的效果。

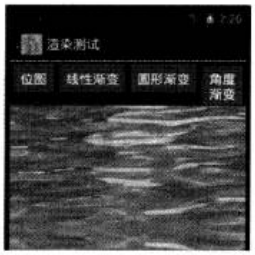


图 7.9　位图平铺的渲染效果

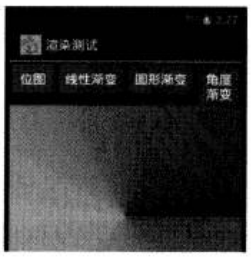


图 7.10　角度渐变的效果

图 7.9、图 7.10 显示了 Android 提供的 BitmapShader、SweepGradient 两种 Shader 的渲染效果，读者可自行运行该程序去查看其他 Shader 的渲染效果。

7.4　逐帧（Frame）动画

逐帧（Frame）是最容易理解的动画，它要求开发者把动画过程的每张静态图片都收集

起来，然后由 Android 来控制依次显示这些静态图片，然后利用人眼“视觉暂留”的原理，给用户造成“动画”的错觉。逐帧动画的动画原理与放电影的原理完全一样。

7.4.1 AnimationDrawable 与逐帧动画

前面介绍定义 Android 资源时已经介绍了动画资源，事实上逐帧动画通常也是采用 XML 资源文件进行定义的。

定义逐帧动画非常简单，只要在<animation-list.../>元素中使用<item.../>子元素定义动画的全部帧，并指定各帧的持续时间即可。定义逐帧动画的语法格式如下：

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
     android:oneshot=["true" | "false"] >
     <item android:drawable="@[package:]drawable/drawable_resource_name"
          android:duration="integer" />
</animation-list>
```

上面的语法格式中 android:onshot 控制该动画是否循环播放，如果该参数指定为 true，则动画将不会循环播放；否则该动画将会循环播放。每个<item.../>子元素添加一帧。



提示：

Android 完全支持在 Java 代码中创建逐帧动画，如果开发者喜欢，开发者完全可以先创建 AnimationDrawable 对象，然后调用 addFrame (Drawable frame, int duration)向该动画中添加帧，每调用一次 addFrame 方法，就向<animation-list.../>元素中添加一个<item.../>子元素。

一旦程序获取了 AnimationDrawable 对象之后，接下来就可用 ImageView 把 AnimationDrawable 显示出来——习惯上把 AnimationDrawable 设成 ImageView 的背景即可。

下面是逐帧动画的简单示例，该程序先使用如下代码定义了一个逐帧动画资源。

程序清单：codes\07\7.4\FatPo\res\anim\fat_po.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 指定动画循环播放 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
     android:oneshot="false">
     <!-- 添加多个帧 -->
     <item android:drawable="@drawable/fat_po_f01" android:duration="60" />
     <item android:drawable="@drawable/fat_po_f02" android:duration="60" />
     <item android:drawable="@drawable/fat_po_f03" android:duration="60" />
     <!-- 下面省略了多个类似的 item 定义 -->
     ...
</animation-list>
```

上面的 fat_po.xml 文件定义了一个逐帧动画资源，接下来就可以在程序中使用 ImageView 来显示该动画。

需要指出的是：AnimationDrawable 代表的动画默认是不播放的，必须在程序中启动动画播放才可以。AnimationDrawable 提供了如下两个方法来开始、停止动画。

- start()：开始播放动画。
- stop()：停止播放动画。

下面的程序中包含两个按钮，一个按钮用于开发动画，另一个按钮可暂停动画。

程序清单：codes\07\7.4\FatPo\src\org\crazyit\image\FatPo.java

```
public class FatPo extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取两个按钮
        Button play = (Button) findViewById(R.id.play);
        Button stop = (Button) findViewById(R.id.stop);
        ImageView imageView = (ImageView) findViewById(R.id.anim);
        // 获取 AnimationDrawable 动画对象
        final AnimationDrawable anim = (AnimationDrawable) imageView
                .getBackground();
        play.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 开始播放动画
                anim.start();
            }
        });
        stop.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 停止播放动画
                anim.stop();
            }
        });
    }
}
```



图 7.11　逐帧动画

上面的程序中两行粗体字代码用于控制动画的播放和停止。运行该程序，单击程序中的“播放”按钮，将可以看到程序中“功夫熊猫”开始表演，如图 7.11 所示。

实例：在指定点爆炸

前面介绍“雷电”飞机游戏时已经提到，当敌机与用户自己的飞机碰撞、或者自己的飞机发射的子弹与敌机碰撞时，都应该在碰撞点播放“飞机爆炸”的动画。

爆炸效果实际上是一个逐帧动画，开发者需要收集从开始爆炸到爆炸结束的所有静态图片，再将这些图片定义成一个逐帧动画，接着在碰撞点播放该逐帧动画即可。

本示例为了突出此处的“主题”——在指定点爆炸，并未增加飞机控制这些细节，只是简单地检测触摸屏事件，当用户触碰触摸屏时，程序将会在触碰点“爆炸”。

该程序先使用如下代码来定义爆炸的逐帧动画资源。

程序清单：codes\07\7.4\Blast\res\anim\blast.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义动画只播放一次，不循环 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="true" >
```

```
    <item android:drawable="@drawable/bom_f01" android:duration="80" />
    <item android:drawable="@drawable/bom_f02" android:duration="80" />
    <!-- 省略其他相似的 item 元素 -->
    ...
</animation-list>
```

接下来就可以利用上面的 blash.xml 所定义的动画资源了——当程序检测到用户触摸屏幕时，程序就在该触摸点播放该动画资源。程序如下。

程序清单：codes\07\7.4\Blast\src\org\crazyit\image\Blast.java

```
public class Blast extends Activity
{
    private MyView myView;
    private AnimationDrawable anim;
    private MediaPlayer bomb;
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 使用 FrameLayout 布局管理器，它允许组件自己控制位置
        FrameLayout frame = new FrameLayout(this);
        setContentView(frame);
        // 设置使用背景
        frame.setBackgroundResource(R.drawable.back);
        // 加载音效
        bomb = MediaPlayer.create(this, R.raw.bomb);
        myView = new MyView(this);
        // 设置 myView 用于显示 blast 动画
        myView.setBackgroundResource(R.anim.blast);
        // 设置 myView 默认为隐藏
        myView.setVisibility(View.INVISIBLE);
        // 获取动画对象
        anim = (AnimationDrawable) myView.getBackground();
        frame.addView(myView);
        frame.setOnTouchListener(new OnTouchListener()
        {
            @Override
            public boolean onTouch(View source, MotionEvent event)
            {
                // 只处理按下事件（避免每次产生两个动画效果）
                if (event.getAction() == MotionEvent.ACTION_DOWN)
                {
                    // 先停止动画播放
                    anim.stop();
                    float x = event.getX();
                    float y = event.getY();
                    // 控制 myView 的显示位置
                    myView.setLocation((int) y - 40, (int) x - 20);
                    myView.setVisibility(View.VISIBLE);
                    // 启动动画
                    anim.start();
                    // 播放音效
                    bomb.start();
                }
                return false;
            }
        });
    }
    // 定义一个自定义 View，该自定义 View 用于播放“爆炸”效果
    class MyView extends ImageView
    {
        public MyView(Context context)
```

```
        {
            super(context);
        }
        // 定义一个方法，该方法用于控制 MyView 的显示位置
        public void setLocation(int top, int left)
        {
            this.setFrame(left, top, left + 40, top + 40);
        }
        // 重写该方法，控制如果动画播放到最后一帧时，隐藏该 View
        @Override
        protected void onDraw(Canvas canvas) // ①
        {
            try
            {
                Field field = AnimationDrawable.class
                        .getDeclaredField("mCurFrame");
                field.setAccessible(true);
                // 获取 anim 动画的当前帧
                int curFrame = field.getInt(anim);
                // 如果已经到了最后一帧
                if (curFrame == anim.getNumberOfFrames() - 1)
                {
                    // 让该 View 隐藏
                    setVisibility(View.INVISIBLE);
                }
            }
            catch (Exception e)
            {
            }
            super.onDraw(canvas);
        }
    }
}
```

上面的程序中第一段粗体字代码就是触摸屏事件的响应代码，该程序把 ImageView 移动到触摸事件的发生点，并播放动画，这就实现在指定点爆炸的效果。

需要指出的是程序中①号代码所定义的方法对于该程序不是必要的——即使程序删除该方法也完全正常，但这只是恰好因为爆炸效果的最后一帧是空白。换一种情况，如果爆炸动画的最后一帧不是空白，而程序又没有控制隐藏播放动画的 ImageView，用户将会看到动画结束了，但动画效果依然残留在那里，为了解决这个问题，就可借助于上面的①号代码所定义的方法：它会自动检测动画播放到最后一帧时隐藏该 ImageView。

7.5 补间（Tween）动画

Android 除了支持逐帧动画之外，也提供了对补间（Tween）动画的支持，补间动画就是指开发者只需指定动画开始、动画结束“关键帧”，而动画变化的“中间帧”由系统计算、并补齐，这也是笔者把 Tween 动画翻译为“补间动画”的原因。

➤➤7.5.1 Tween 动画与 Interpolator

有过 Flash 设计经验的人应该对补间动画有很好的概念。就笔者的经验来看，Flash 的补间动画支持比 Android 的更简单、功能更强。对于没有 Flash 设计经验的读者，图 7.12 可作

为补间动画的示意。



图 7.12 补间动画的示意

从图 7.12 可以看出，对于补间动画而言，开发者无须“逐一”定义动画过程中的每一帧，他只要定义动画开始、结束的关键帧，并指定动画的持续时间即可。

从图 7.12 可以看出，补间动画所定义的开始帧、结束帧其实只是一些简单的变化，比如图形大小的缩放、旋转角度的改变等。Android 使用 Animation 代表抽象的动画类，它包括如下几个子类。

- AlphaAnimation：透明度改变的动画。创建该动画时要指定动画开始时的透明度、结束时的透明度和动画持续时间。其中透明度可从 0 变化到 1。
- ScaleAnimation：大小缩放的动画。创建该动画时要指定动画开始时的缩放比（以 *X*、*Y* 轴的缩放参数来表示）、结束时动画的缩放比（以 *X*、*Y* 轴的缩放参数来表示），并指定动画持续时间。由于缩放时以不同点为中心时缩放效果并不相同，因此指定缩放动画时还要通过 pivotX、pivotY 来指定“缩放中心”的坐标。
- TranslateAnimation：位移变化的动画，创建该动画时只要指定动画开始时的位置（以 *X*、*Y* 坐标来表示）、结束时的位置（以 *X*、*Y* 坐标来表示），并指定动画持续时间即可。
- RotateAnimation：旋转动画，创建该动画时只要指定动画开始时的旋转角度、结束时的旋转角度，并指定动画持续时间即可。由于旋转时以不同点为中心时旋转效果并不相同，因此指定旋转动画时还要通过 pivotX、pivotY 来指定“旋转轴心”的坐标。

一旦为补间动画指定了三个必要信息，Android 就会根据动画的开始帧、结束帧、动画持续时间计算出需要在中间“补入”多少帧，并计算所有补入帧的图形。当用户浏览补间动画时，他眼中看到的依然是“逐帧动画”。

为了控制在动画期间需要动态“补入”多少帧，具体在动画运行的哪些时刻补入帧，需要借助于 Interpolator。



提示：

Interpolator 在笔者以前念大学时看到有资料将它翻译为“插值”，这个翻译基本还可以：补间动画定义的是动画开始、结束的关键帧，Android 需要在开始帧、结束帧之间动态地计算、插入大量的帧，而 Interpolator 用于控制“插入帧”的行为，因此翻译为插值也是合适的；现在也有些资料将 Interpolator 翻译得比较生僻，难以理解。为避免读者对各种翻译感到疑惑，本书后面笔者一律使用英文单词 Interpolator，不会去强行将它翻译成中文。

Interpolator 根据特定算法计算出整个动画所需要动态插入帧的密度和位置，简单地说，Interpolator 负责控制动画的变化速度，这就使得基本的动画效果（Alpha、Scale、Translate、

Rotate）能以匀速变化、加速、减速、抛物线速度等各种速度变化。

Interpolator 是一个接口，它定义了所有 Interpolator 都需要实现的方法：float getInterpolation(float input)，开发者完全可以通过实现 Interpolator 来控制动画的变化速度。

Android 为 Interpolator 提供了如下几个实现类，分别用于实现不同动画变化速度。

- LinearInterpolator：动画以均匀的速度改变。
- AccelerateInterpolator：在动画开始的地方改变速度较慢，然后开始加速。
- AccelerateDecelerateInterpolator：在动画开始、结束的地方改变速度较慢，在中间的时候加速。
- CycleInterpolator：动画循环播放特定的次数，变化速度按正弦曲线改变。
- DecelerateInterpolator：在动画开始的地方改变速度较快，然后开始减速。

为了在动画资源文件中指定补间动画所使用的 Interceptor，定义补间动画的<set.../>元素支持一个 android:interpolator 属性，该属性的属性值可指定为 Android 默认支持的 Interceptor。例如：

- @android:anim/linear_interpolator
- @android:anim/accelerate_interpolator
- @android:anim/accelerate_decelerate_interpolator

……

其实上面的写法很有规律，它们就是把系统提供的 Interpolator 实现类的类名的驼峰写法改为中划线写法即可。

一旦在程序中通过 AnimationUtils 得到代表补间动画的 Animation 之后，接下来就可调用 View 的 startAnimation(Animation anim)方法开始对该 View 执行动画了。

7.5.2 位置、大小、旋转度、透明度改变的补间动画

虽然 Android 允许在程序中创建 Animation 对象，但实际上一般都会采用动画资源文件来定义补间动画，前面已经介绍过定义补间动画资源文件的格式，此处不再赘述。

下面以一个示例来介绍补间动画，该示例包括了两个动画资源文件，第一个动画资源文件控制图片以旋转的方式缩小，该动画资源文件如下。

程序清单：codes\07\7.5\TweenAnim\res\anim\anim.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 指定动画匀速改变 -->
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/linear_interpolator">
    <!-- 定义缩放变换 -->
    <scale android:fromXScale="1.0"
        android:toXScale="0.01"
        android:fromYScale="1.0"
        android:toYScale="0.01"
        android:pivotX="50%"
        android:pivotY="50%"
        android:fillAfter="true"
        android:duration="3000"/>
    <!-- 定义透明度的变换 -->
    <alpha
        android:fromAlpha="1"
        android:toAlpha="0.05"
        android:duration="3000"/>
```

```
        <!-- 定义旋转变换 -->
        <rotate
              android:fromDegrees="0"
              android:toDegrees="1800"
              android:pivotX="50%"
              android:pivotY="50%"
              android:duration="3000"/>
</set>
```

上面的动画资源指定动画匀速变化，同时进行缩放、透明度改变、旋转三种改变，动画持续时间为三秒。

第二个动画资源则控制图片以动画的方式恢复回来，对应的动画资源如下。

程序清单：codes\07\7.5\TweenAnim\res\anim\reverse.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 指定动画匀速改变 -->
<set xmlns:android="http://schemas.android.com/apk/res/android"
      android:interpolator="@android:anim/linear_interpolator"
      android:startOffset="3000">
      <!-- 定义缩放变换 -->
      <scale android:fromXScale="0.01"
            android:toXScale="1"
            android:fromYScale="0.01"
            android:toYScale="1"
            android:pivotX="50%"
            android:pivotY="50%"
            android:fillAfter="true"
            android:duration="3000"/>
      <!-- 定义透明度的变换 -->
      <alpha
            android:fromAlpha="0.05"
            android:toAlpha="1"
            android:duration="3000"/>
      <!-- 定义旋转变换 -->
      <rotate
            android:fromDegrees="1800"
            android:toDegrees="0"
            android:pivotX="50%"
            android:pivotY="50%"
            android:duration="3000"/>
</set>
```

定义动画资源之后，接下来就可以利用 AnimationUtils 工具类来加载指定的动画资源，加载成功后会返回一个 Animation，该对象即可控制图片或视图播放动画。

下面的程序将会负责加载动画资源，并使用 Animation 来控制图片播放动画。

程序清单：codes\07\7.5\TweenAnim\src\org\crazyit\image\TweenAnim.java

```
public class TweenAnim extends Activity
{
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            final ImageView flower = (ImageView)
                  findViewById(R.id.flower);
            // 加载第一份动画资源
            final Animation anim = AnimationUtils
                  .loadAnimation(this, R.anim.anim);
```

```
        // 设置动画结束后保留结束状态
        anim.setFillAfter(true);
        // 加载第二份动画资源
        final Animation reverse = AnimationUtils.loadAnimation(this
            , R.anim.reverse);
        // 设置动画结束后保留结束状态
        reverse.setFillAfter(true);
        Button bn = (Button) findViewById(R.id.bn);
        final Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                if (msg.what == 0x123)
                {
                    flower.startAnimation(reverse);
                }
            }
        };
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                flower.startAnimation(anim);
                // 设置 3.5 秒后启动第二个动画
                new Timer().schedule(new TimerTask()
                {
                    @Override
                    public void run()
                    {
                        handler.sendEmptyMessage(0x123);
                    }
                }, 3500);
            }
        });
    }
}
```

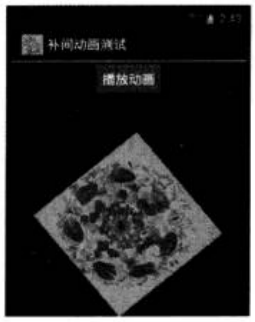


图 7.13　补间动画

正如上面两行粗体字代码所看到的，当用户单击程序中指定按钮时，程序对 flower 图片播放第一个动画；程序使用定时器设置 3.5 秒后对 flower 图片播放第二个动画。运行该程序，单击按钮将看到程序中间的图片先旋转着缩小、变淡，然后旋转着放大、透明度也逐渐恢复正常，如图 7.13 所示。

实例：蝴蝶飞舞

很多实际的动画往往同时运行两个动画，比如我们要做一个小游戏：需要让用户控制游戏中的主角移动——当主角移动时，不仅要控制它的位置改变，还应该在它移动时播放逐帧动画来让用户感觉更“逼真”。

下面一个实例将会结合逐帧动画和补间动画来开发一个“蝴蝶飞舞”的效果，在这个实例中，蝴蝶飞行时的振翅效果是逐帧动画；蝴蝶飞行时的位置改变是补间动画。

先为该实例定义如下动画资源。

程序清单：codes\07\7.5\butterfly\res\anim\butterfly.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!-- 定义动画循环播放 -->
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
      android:oneshot="false">
      <item android:drawable="@drawable/butterfly_f01" android:duration="120" />
      <item android:drawable="@drawable/butterfly_f02" android:duration="120" />
      <!-- 下面省略了相似的 item 元素 -->
      ...
</animation-list>
```

定义了上面逐帧动画的动画资源后，接下来在程序中使用一个 ImageView 显示该动画资源即可，这就可以看到蝴蝶“振翅”的效果了。由于蝴蝶飞舞主要是位移改变，接下来可以在程序中通过 TranslateAnimation 以动画的方式改变 ImageView 的位置，这样就可实现“蝴蝶飞舞”的效果了，程序如下。

程序清单：codes\07\7.5\butterfly\src\org\crazyit\image\Butterfly.java

```
public class Butterfly extends Activity
{
      // 记录蝴蝶 ImageView 当前的位置
      private float curX = 0;
      private float curY = 30;
      // 记录蝴蝶 ImageView 下一个位置的坐标
      float nextX = 0;
      float nextY = 0;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取显示蝴蝶的 ImageView 组件
            final ImageView imageView = (ImageView)
                  findViewById(R.id.butterfly);
            final Handler handler = new Handler()
            {
                  @Override
                  public void handleMessage(Message msg)
                  {
                        if (msg.what == 0x123)
                        {
                              // 横向上一直向右飞
                              if (nextX > 320)
                              {
                                    curX = nextX = 0;
                              }
                              else
                              {
                                    nextX += 8;
                              }
                              // 纵向上可以随机上下
                              nextY = curY + (float) (Math.random() * 10 - 5);
                              // 设置显示蝴蝶的 ImageView 发生位移改变
                              TranslateAnimation anim = new TranslateAnimation(
                                    curX, nextX, curY, nextY);
                              curX = nextX;
                              curY = nextY;
                              anim.setDuration(200);
                              // 开始位移动画
                              imageView.startAnimation(anim); // ①
                        }
                  }
            };
```

```
        final AnimationDrawable butterfly = (AnimationDrawable)
                imageView.getBackground();
        imageView.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 开始播放蝴蝶振翅的逐帧动画
                butterfly.start();// ②
                // 通过定制器控制每 0.2 秒运行一次 TranslateAnimation 动画
                new Timer().schedule(new TimerTask()
                {
                    @Override
                    public void run()
                    {
                        handler.sendEmptyMessage(0x123);
                    }
                }, 0, 200);
            }
        });
    }
}
```



图 7.14　蝴蝶飞舞

上面的程序中①号粗体字代码位于 Handler 的消息处理方法内，这样程序每隔 0.2 秒即对该 ImageView 执行一次位移动画；程序的②号粗体字代码则用于播放 butterfly 动画（蝴蝶振翅效果）。运行上面的程序，单击蝴蝶，即可看到屏幕有只蝴蝶从左向右飞舞，如图 7.14 所示。

7.5.3　自定义补间动画

Android 提供了 Animation 作为补间动画抽象基类，而且为该抽象基类提供了 AlphaAnimation、RotateAnimation、ScaleAnimation、TranslateAnimation 四个实现类，这四个实现类只是补间动画的四种基本形式：透明度改变、旋转、缩放、位移，在实际项目中可能还需要一些更复杂的动画，比如让图片在“三维”空间内进行旋转动画等，这就需要开发者自己开发补间动画了。

自定义补间动画并不难，需要继承 Animation，继承 Animation 时关键是要重写该抽象基类的 applyTransformation(float interpolatedTime, Transformation t)方法，该方法中两个参数的说明如下。

- interpolatedTime：代表了动画的时间进行比。不管动画实际的持续时间如何，当动画播放时，该参数总是自动从 0 变化到 1 的。
- Transformation：该参数代表了补间动画在不同时刻对图形或组件的变形程度。

从上面的介绍可以看出，实现自定义动画的关键就在于重写 applyTransformation 方法时根据 interpolatedTime 时间来动态地计算动画对图片或视图的变形程度。

Transformation 代表了对图片或视图的变形，该对象里封装了一个 Matrix 对象，对它所包装了 Matrix 进行位移、倾斜、旋转等变换时，Transformation 将会控制对应的图片或视图进行相应的变换。

为了控制图片或 View 进行三维空间的变换，还需要借助于 Android 提供的一个 Camera，这个 Camera 并非代表手机的摄像头，只是一个空间变换工具，作用有点类似于 Matrix，但功能更强大。

Camera 提供了如下常用的方法。

- getMatrix(Matrix matrix)：将 Camera 所做的变换应用到指定 matrix 上。
- rotateX(float deg)：将目标组件沿 *X* 轴旋转。
- rotateY(float deg)：将目标组件沿 *Y* 轴旋转。
- rotateZ(float deg)：将目标组件沿 *Z* 轴旋转。
- translate(float x, float y, float z)：把目标组件在三维空间里进行位移变换。
- applyToCanvas(Canvas canvas)：把 Camera 所做的变换应用到 Canvas 上。

从上面的方法可以看出，Camera 的主要用于支持三维空间的变换，那么手机中三维空间的坐标系统是怎样的呢？图 7.15 显示了手机屏幕上的三维坐标系统。

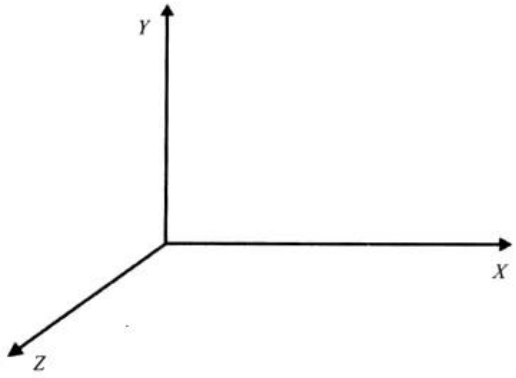


图 7.15 手机屏幕上的三维坐标

当 Camera 控制图片或 View 沿 *X*、*Y* 或 *Z* 轴旋转时，被旋转的图片或 View 将会呈现出三维透视的效果。图 7.16 显示了一张图片沿着 *Y* 轴旋转的效果。

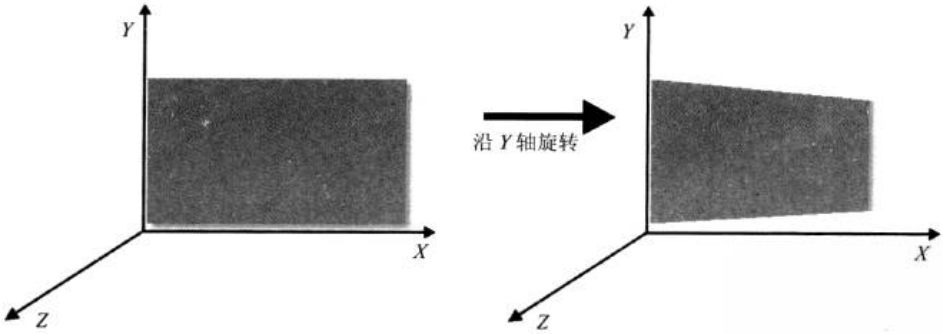


图 7.16 三维空间内沿 *Y* 轴旋转的示意

关于三维透视理论的知识，此处限于篇幅不便深入解释，如果读者有 3ds Max 或 Maya 之类的设计经验，应该很容易理解三维透视的基本理论，否则建议读者自行阅读相关内容。

下面程序将会利用 Camera 来自定义在三维空间的动画，该程序的自定义动画类的代码如下。

程序清单：\codes\07\7.5\ListViewTween\src\org\crazyit\image\MyAnimation.java

```
public class MyAnimation extends Animation
{
    private float centerX;
    private float centerY;
    // 定义动画的持续事件
```

```
        private int duration;
        private Camera camera = new Camera();
        public MyAnimation(float x, float y, int duration)
        {
            this.centerX = x;
            this.centerY = y;
            this.duration = duration;
        }
        @Override
        public void initialize(int width, int height
            , int parentWidth, int parentHeight)
        {
            super.initialize(width, height, parentWidth, parentHeight);
            // 设置动画的持续时间
            setDuration(duration);
            // 设置动画结束后效果保留
            setFillAfter(true);
            setInterpolator(new LinearInterpolator());
        }
        /*
         * 该方法的 interpolatedTime 代表了抽象的动画持续时间，不管动画实际持续时间多长，
         * interpolatedTime 参数总是从 0（动画开始时）～1（动画结束时）
         * Transformation 参数代表了对目标组件所做的改变.
         */
        @Override
        protected void applyTransformation(float interpolatedTime
            , Transformation t)
        {
            camera.save();
            // 根据 interpolatedTime 时间来控制 X、Y、Z 上的偏移
            camera.translate(100.0f - 100.0f * interpolatedTime,
                    150.0f * interpolatedTime - 150,
                    80.0f - 80.0f * interpolatedTime);
            // 设置根据 interpolatedTime 时间在 Y 轴上旋转不同角度
            camera.rotateY(360 * (interpolatedTime));
            // 设置根据 interpolatedTime 时间在 X 轴上旋转不同角度
            camera.rotateX((360 * interpolatedTime));
            // 获取 Transformation 参数的 Matrix 对象
            Matrix matrix = t.getMatrix();
            camera.getMatrix(matrix);
            matrix.preTranslate(-centerX, -centerY);
            matrix.postTranslate(centerX, centerY);
            camera.restore();
        }
    }
```

上面的程序中自定义动画的关键就是两行粗体字代码，这两行代码将会根据动画的进行时间来控制 View 在 *X* 轴、*Y* 轴上的旋转——这就会具有在三维空间内旋转的效果。

提供了上面的自定义动画类之后，接下来既可用该自定义动画来控制图片，也可控制 View 组件，因为动画就是通过调用 View 的 startAnimation(Animation anim)来启动的。下面是使用 MyAnimation 进行动画的程序代码。

程序清单：codes\07\7.5\ListViewTween\src\org\crazyit\image\ListViewTween.java

```
public class ListViewTween extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        // 获取 ListView 组件
        ListView list = (ListView) findViewById(R.id.list);
        WindowManager windowManager = (WindowManager)
            getSystemService(WINDOW_SERVICE);
        Display display = windowManager.getDefaultDisplay();
        DisplayMetrics metrice = new DisplayMetrics();
        // 获取屏幕的宽和高
        display.getMetrics(metrice);
        // 设置对 ListView 组件应用动画
        list.setAnimation(new MyAnimation(metrice.xdpi / 2
            , metrice.ydpi / 2, 3500));
    }
}
```

上面的程序中粗体字代码对 ListView 使用 MyAnimation 播放动画，这意味着程序中 ListView 将会以三维变换的动画方式出现，如图 7.17 所示。



图 7.17 三维空间变换的动画

7.6 属性动画

前面介绍 Android 资源时已经提到了属性动画，从某种角度来看，属性动画是增强版的补间动画，属性动画的强大可以体现在如下两方面。

- 补间动画只能定义两个关键帧在“透明度”、“旋转”、“倾斜”、“位移”4 个方面的变化，但属性动画可以定义任何属性的变化。
- 补间动画只能对 UI 组件执行动画，但属性动画几乎可以对任何对象执行动画（不管它是否显示在屏幕上）。

与补间动画类似的是，属性动画也需要定义如下几方面的属性。

- 动画持续时间。该属性的默认值是 300 毫秒。在属性动画资源文件中通过 android:duration 属性指定。
- 动画差值方式。该属性的作用与补间动画中插值属性的作用基本类似。在属性动画资源文件中通过 android:interpolator 属性指定。
- 动画重复次数。指定动画重复播放的次数。在属性动画资源文件中通过 android:repeatCount 属性指定。
- 重复行为。指定动画播放结束后、重复下次动画时，是从开始帧再次播放到结束帧，还是从结束帧反向播放到开始帧。在属性动画资源文件中通过 android:repeatMode 属性指定。
- 动画集。开发者可以将多个属性动画合并成一组，既可让这组属性动画按次序播放，也可让这组属性动画同时播放。在属性动画资源文件中通过<set..../>元素来组合，该元素的 android:ordering 属性指定该组动画是按次序播放，还是同时播放。
- 帧刷新频率。指定每隔多长时间播放一帧。该属性默认值为 10 毫秒。

7.6.1 属性动画的 API

属性动画共涉及如下 API。

- Animator：它提供了创建属性动画的基类，基本上不会直接地使用该类。通常该类

只用于被继承并重写它的相关方法。

- ValueAnimator：属性动画主要的时间引擎，它负责计算各个帧的属性值。它定义了属性动画的绝大部分核心功能，包括计算各帧的相关属性值，负责处理更新事件，按属性值的类型控制计算规则。属性动画主要由两方面组成：① 计算各帧的相关属性值；② 为指定对象设置这些计算后的值。ValueAnimator 只负责第一方面内容，因此程序员必须根据 ValueAnimator 计算并监听值更新来更新对象的相关属性值。
- ObjectAnimator：它是 ValueAnimator 子类，允许程序员对指定对象的属性执行动画。实际应用中，ObjectAnimator 使用起来更加简单，因此更加常用。在少数场景下，由于 ObjectAnimator 存在一些限制，可能需要考虑使用 ValueAnimator。
- AnimatorSet：它是 Animator 的子类，用于组合多个 Animator，并指定多个 Animator 是按次序播放，还是同时播放。

除此之外，属性动画还需要利用一个 Evaluator（计算器），该工具类控制属性动画如何计算属性值。Android 提供了如下 Evaluator。

- IntEvaluator：用于计算 int 类型属性值的计算器。
- FloatEvaluator：用于计算 float 类型属性值的计算器。
- ArgbEvaluator：用于计算以十六进制形式表示的颜色值的计算器。
- TypeEvaluator：它是计算器接口，开发者可以通过实现该接口来实现自定义计算器。如果需要对 int,float 或者颜色值以外类型的属性执行属性动画，可能需要实现 TypeEvaluator 接口来实现定义计算器。

1. 使用 ValueAnimator 创建动画

使用 ValueAnimator 动画可按如下 4 个步骤。

① 调用 ValueAnimator 的 ofInt()、ofFloat()或 ofObject()静态方法创建 ValueAnimator 实例。

② 调用 ValueAnimator 的 setXxx()设置动画持续时间、插值方式、重复次数等。

③ 调用 ValueAnimator 的 start()方法启动动画。

④ 为 ValueAnimator 注册 AnimatorUpdateListener 监听器，在该监听器中可以监听 ValueAnimator 计算出来的值的改变，并将这些值应用到指定对象。

例如如下代码片段：

```
ValueAnimator animation = ValueAnimator.ofFloat(0f, 1f);
animation.setDuration(1000);
animation.start();
```

上面的例子实现了在 1000ms 内，值从 0～1 的变化。

除此之外，开发者也可以提供一个自定义的 Evaluator 计算器，例如如下代码：

```
ValueAnimator animation = ValueAnimator.ofObject(new MyTypeEvaluator(), startVal,
endVal)
animation.setDuration(1000);
animation.start();
```

上面的代码片段中，ValueAnimator 仅仅是计算动画过程中变化的值，并没有把这些计算出来的值应用到任何对象上，因此也不会显示任何动画。

如果希望使用 ValueAnimator 创建动画，还需要注册一个监听器：AnimatorUpdateListener，该监听器负责更新对象的属性值。在实现这个监听器的时候，可以通过 getAnimatedValue()

的方法来获取当前帧的值，并将该计算出来的值应用到指定对象上。当该对象的属性持续改变时，该对象也就呈现出动画效果。

2. 使用 ObjectAnimator 创建动画

ObjectAnimator 继承了 ValueAnimator，因此它可以直接将 ValueAnimator 在动画过程中计算出来的值应用到指定对象的指定属性上（ValueAnimator 则需要注册一个监听器来完成这个工作）。因此使用 ObjectAnimator 就不需要注册 AnimatorUpdateListener 监听器。

使用 ObjectAnimator 的 ofInt()、ofFloat()或 ofObject()静态方法创建 ObjectAnimator 时，需要指定具体的对象，以及对象的属性名。

例如如下代码片段：

```
ObjectAnimator anim = ObjectAnimator.ofFloat(foo, "alpha", 0f, 1f);
anim.setDuration(1000);
anim.start();
```

与 ValueAnimator 不同的是，使用 ObjectAnimator 有如下几个注意点。

- 要为该对象对应的属性提供 setter 方法，如上例中需要为 foo 对象提供 setAlpha(float value)方法。
- 如果调用 ObjectAnimator 的 ofInt()、ofFloat()或 ofObject()工厂方法时 values... 参数只提供了一个值（本来需要提供开始值和结束值），那么该值会被认为是结束值。那么该对象应该为该属性提供一个 getter 方法，该 getter 方法的返回值将被作为开始值。
- 如果动画的对象是 View，为了能显示动画效果，可能还需要在 onAnimationUpdate()事件监听方法中调用 View.invalidate()方法来刷新屏幕的显示，比如对 Drawable 对象的 color 属性执行动画。但 View 定义的 setter 方法，如 setAlpha()和 setTranslationX()等方法，都会自动地调用 invalidate()方法，因此不需要额外地调用 invalidate()方法。

7.6.2 使用属性动画

属性动画既可作用于 UI 组件，也可作用于普通的对象（即使它没有在 UI 界面上绘制出来）。

定义属性动画有如下两种方式。

- 使用 ValueAnimator 或 ObjectAnimator 的静态工厂方法来创建动画。
- 使用资源文件来定义动画。

使用属性动画的步骤如下：

① 创建 ValueAnimator 或 ObjectAnimator 对象——既可从 XML 资源文件加载该动画资源，也可直接调用 ValueAnimator 或 ObjectAnimator 的静态工厂方法来创建动画。

② 根据需要为 Animator 对象设置属性。

③ 如果需要监听 Animator 的动画开始事件、动画结束事件、动画重复事件、动画值改变事件，并根据事件提供响应处理代码，应该为 Animator 对象设置事件监听器。

④ 如果有多个动画需要按次序或同时播放，应使用 AnimatorSet 组合这些动画。

⑤ 调用 Animator 对象的 start()方法启动动画。

下面的示例示范了如何利用属性动画来控制“小球”掉落动画，该示例会监听用户在屏

幕上的“触屏”时间，程序会在屏幕的触摸点绘制一个小球，并用动画控制该小球向下掉落。

该示例的界面布局文件非常简单，界面布局文件中只有一个 LinearLayout，因此此处不再给出界面布局文件。下面是该示例的 Activity 代码。

程序清单：codes\07\7.6\AnimatorTest\src\org\crazyit\image\AnimatorTest.java

```
public class AnimatorTest extends Activity
{
    // 定义小球的大小的常量
    static final float BALL_SIZE = 50F;
    // 定义小球从屏幕上方下落到屏幕底端的总时间
    static final float FULL_TIME = 1000;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout container = (LinearLayout)
            findViewById(R.id.container);
        // 设置该窗口显示 MyAnimationView 组件
        container.addView(new MyAnimationView(this));
    }
    public class MyAnimationView extends View
        implements AnimatorUpdateListener
    {
        public final ArrayList<ShapeHolder> balls
            = new ArrayList<ShapeHolder>();
        public MyAnimationView(Context context)
        {
            super(context);
            setBackgroundColor(Color.WHITE);
        }
        @Override
        public boolean onTouchEvent(MotionEvent event)
        {
            // 如果触碰事件不是按下、移动事件
            if (event.getAction() != MotionEvent.ACTION_DOWN
                && event.getAction() != MotionEvent.ACTION_MOVE)
            {
                return false;
            }
            //  在事件发生点添加一个小球（用一个圆形代表）
            ShapeHolder newBall = addBall(event.getX(), event.getY());
            // 计算小球下落动画开始时的 y 坐标
            float startY = newBall.getY();
            // 计算小球下落动画结束时的 y 坐标（落到屏幕最下方，就是屏幕高度减去小球高度）
            float endY = getHeight() - BALL_SIZE;
            // 获取屏幕高度
            float h = (float) getHeight();
            float eventY = event.getY();
            // 计算动画的持续时间
            int duration = (int) (FULL_TIME * ((h - eventY) / h));
            // 定义小球“落下”的动画：让 newBall 对象的 y 属性从事件发生点变化到屏幕最下方
            ValueAnimator fallAnim = ObjectAnimator.ofFloat(
                newBall, "y", startY, endY);
            // 设置 fallAnim 动画的持续时间
            fallAnim.setDuration(duration);
            // 设置 fallAnim 动画的插值方式：加速插值
            fallAnim.setInterpolator(new AccelerateInterpolator());
            // 为 fallAnim 动画添加监听器
            // 当 ValueAnimator 的属性值发生改变时，将会激发该监听器的事件监听方法
```

```
        fallAnim.addUpdateListener(this);
        // 定义对 newBall 对象的 alpha 属性执行从 1 到 0 的动画（即定义渐隐动画）
        ObjectAnimator fadeAnim = ObjectAnimator.ofFloat(newBall
            , "alpha", 1f, 0f);
        // 设置动画持续时间
        fadeAnim.setDuration(250);
        // 为 fadeAnim 动画添加监听器
        fadeAnim.addListener(new AnimatorListenerAdapter()
        {
            // 当动画结束时
            @Override
            public void onAnimationEnd(Animator animation)
            {
                // 动画结束时将该动画关联的 ShapeHolder 删除
                balls.remove(((ObjectAnimator) animation).getTarget());
            }
        });
        // 为 fadeAnim 动画添加监听器
        // 当 ValueAnimator 的属性值发生改变时，将会激发该监听器的事件监听方法
        fadeAnim.addUpdateListener(this);
        // 定义一个 AnimatorSet 来组合动画
        AnimatorSet animatorSet = new AnimatorSet();
        // 指定在播放 fadeAnim 之前，先播放 bouncer 动画
        animatorSet.play(fallAnim).before(fadeAnim);
        // 开发播放动画
        animatorSet.start();
        return true;
    }
    private ShapeHolder addBall(float x, float y)
    {
        // 创建一个圆
        OvalShape circle = new OvalShape();
        // 设置该椭圆的宽、高
        circle.resize(BALL_SIZE, BALL_SIZE);
        // 将圆包装成 Drawable 对象
        ShapeDrawable drawable = new ShapeDrawable(circle);
        // 创建一个 ShapeHolder 对象
        ShapeHolder shapeHolder = new ShapeHolder(drawable);
        // 设置 ShapeHolder 的 x、y 坐标
        shapeHolder.setX(x - BALL_SIZE / 2);
        shapeHolder.setY(y - BALL_SIZE / 2);
        int red = (int) (Math.random() * 255);
        int green = (int) (Math.random() * 255);
        int blue = (int) (Math.random() * 255);
        // 将 red、green、blue 三个随机数组合成 ARGB 颜色
        int color = 0xff000000 + red << 16 | green << 8 | blue;
        // 获取 drawable 上关联的画笔
        Paint paint = drawable.getPaint();
        // 将 red、green、blue 三个随机数除以 4 得到商值组合成 ARGB 颜色
        int darkColor = 0xff000000 | red / 4 << 16
            | green / 4 << 8 | blue / 4;
        // 创建圆形渐变
        RadialGradient gradient = new RadialGradient(
            37.5f, 12.5f, BALL_SIZE, color, darkColor
            , Shader.TileMode.CLAMP);
        paint.setShader(gradient);
        // 为 shapeHolder 设置 paint 画笔
        shapeHolder.setPaint(paint);
        balls.add(shapeHolder);
        return shapeHolder;
    }
```

```
        @Override
        protected void onDraw(Canvas canvas)
        {
            // 遍历 balls 集合中的每个 ShapeHolder 对象
            for (ShapeHolder shapeHolder : balls)
            {
                // 保存 canvas 的当前坐标系统
                canvas.save();
                // 坐标变换：将画布坐标系统平移到 shapeHolder 的 X、Y 坐标处
                canvas.translate(shapeHolder.getX()
                        , shapeHolder.getY());
                // 将 shapeHolder 持有的圆形绘制在 Canvas 上
                shapeHolder.getShape().draw(canvas);
                // 恢复 Canvas 坐标系统
                canvas.restore();
            }
        }
        @Override
        public void onAnimationUpdate(ValueAnimator animation)
        {
            // 指定重绘该界面
            this.invalidate(); //①
        }
    }
}
```

上面的代码中第一段粗体字代码创建了两个 ObjectAnimator 对象，其中第一个 ObjectAnimator 控制该小球的“下落”，第二个 ObjectAnimator 则用于控制该小球的以“渐隐”的方式隐藏——程序为 fadeAnim 绑定了事件监听器，当 fadeAnim 动画播放完成时，程序将小球删除。

上面程序的属性动画并没有控制 UI 组件，而是对一个自定义 ShapeHolder 对象起作用，为了能让 UI 界面不断刷新，看到 UI 界面上小球的动画效果，程序为两个动画都绑定了 AnimatorUpdateListener 监听器，该监听器用于实现当目标对象（小球）的属性发生改变时，该组件会重绘界面——如以上程序中①号代码所示。

上面的程序中第二段粗体字代码用于创建一个圆形渐变，创建圆形渐变的起始颜色的 red、green、blue 值都是随机的，结束颜色是起始颜色值的四分之一。读者可能对程序代码计算颜色值的代码有点疑惑：

```
int color = 0xff000000 + red << 16 | green << 8 | blue;
```

上面的代码其实很简单，这些代码解释如下。

- **0xff000000**：代表了透明度为 **ff**，也就是完全不透明。
- **red** 代表一个 0～255（0～ff）的随机整数，但这个整数要添加 **0xff00000** 中加粗的两个“位”上，也就是要将 **red** 的值左移（16 位，对应为十六进制数的 4 位），这就是 **red<<16** 的原因。
- **green** 代表一个 0～255（0～ff）的随机整数，但这个整数要添加到 **0xff00000** 中加粗的两个“位”上，也就是要将 **green** 的值左移（8 位，对应为十六进制数的 2 位），这就是 **red<<8** 的原因。
- **blue** 代表一个 0～255（0～ff）的随机整数，但这个整数要添加到 **0xff00000** 中加粗的 2 个“位”上，因此 **blue** 值就不需要位移了。

将 red、green、blue 位移后的结果加起来就得到了实际的颜色值，但为了更好的计算性

能，本代码直接使用按位或（|）来累加这些值。

上面的实例还用到了一个 ShapeHolder 类，该类只是负责包装 ShapeDrawable 对象，并为 x、y、width、height、alpha 等属性提供 setter、getter 方法，方便 ObjectAnimator 动画控制它。下面是 ShapeHolder 类的代码。

程序清单：codes\07\7.6\AnimatorTest\src\org\crazyit\image\ShapeHolder.java

```
public class ShapeHolder
{
    private float x = 0, y = 0;
    private ShapeDrawable shape;
    private int color;
    private RadialGradient gradient;
    private float alpha = 1f;
    private Paint paint;
    public ShapeHolder(ShapeDrawable s)
    {
        shape = s;
    }
    // 省略各属性的 setter 和 getter 方法
    ...
}
```

运行该程序，用户在屏幕上拖动手指时将可以看到如图 7.18 所示小球下落的动画。



图 7.18 小球下落

如果为小球增加更多动画，让该小球落到底端时产生弹跳动画，并且再次弹起来，则可以让该示例更加完善。

实例：大珠小珠落玉盘

该实例是对上一个实例的改进，主要是为小球增加了几个动画，控制小球落到底端时小球被压扁，小球会再次弹起，这样就可以开发出“大珠小珠落玉盘”的弹跳动画。

该实例的 Activity 代码如下。

程序清单：codes\07\7.6\BouncingBalls\src\org\crazyit\image\BouncingBalls.java

```
public class BouncingBalls extends Activity
{
    // 定义小球的大小的常量
    static final float BALL_SIZE = 50F;
    // 定义小球从屏幕上方下落到屏幕底端的总时间
    static final float FULL_TIME = 1000;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LinearLayout container = (LinearLayout)
            findViewById(R.id.container);
        // 设置该窗口显示 MyAnimationView 组件
        container.addView(new MyAnimationView(this));
    }
    public class MyAnimationView extends View
    {
        public final ArrayList<ShapeHolder> balls
            = new ArrayList<ShapeHolder>();
```

```
public MyAnimationView(Context context)
{
    super(context);
    // 加载动画资源
    ObjectAnimator colorAnim = (ObjectAnimator) AnimatorInflater
        .loadAnimator(BouncingBalls.this, R.animator.color_anim);
    colorAnim.setEvaluator(new ArgbEvaluator());
    // 对该 View 本身应用属性动画
    colorAnim.setTarget(this);
    // 开始指定动画
    colorAnim.start();
}
@Override
public boolean onTouchEvent(MotionEvent event)
{
    // 如果触碰事件不是按下、移动事件
    if (event.getAction() != MotionEvent.ACTION_DOWN
        && event.getAction() != MotionEvent.ACTION_MOVE)
    {
        return false;
    }
    //  在事件发生点添加一个小球（用一个圆形代表）
    ShapeHolder newBall = addBall(event.getX(), event.getY());
    // 计算小球下落动画开始时的 y 坐标
    float startY = newBall.getY();
    // 计算小球下落动画结束时的 y 坐标（落到屏幕最下方，就是屏幕高度减去小球高度）
    float endY = getHeight() - BALL_SIZE;
    // 获取屏幕高度
    float h = (float) getHeight();
    float eventY = event.getY();
    // 计算动画的持续时间
    int duration = (int) (FULL_TIME * ((h - eventY) / h));
    // 定义小球“落下”的动画：让 newBall 对象的 y 属性从事件发生点变化到屏幕最下方
    ValueAnimator fallAnim = ObjectAnimator.ofFloat(
        newBall, "y", startY, endY);
    // 设置 fallAnim 动画的持续时间
    fallAnim.setDuration(duration);
    // 设置 fallAnim 动画的插值方式：加速插值
    fallAnim.setInterpolator(new AccelerateInterpolator());
    // 定义小球“压扁”的动画：该动画控制小球的 x 坐标“向左移”半个球
    ValueAnimator squashAnim1 = ObjectAnimator.ofFloat(newBall
        , "x", newBall.getX(), newBall.getX() - BALL_SIZE / 2);
    // 设置 squashAnim1 动画持续时间
    squashAnim1.setDuration(duration / 4);
    // 设置 squashAnim1 动画重复 1 次
    squashAnim1.setRepeatCount(1);
    // 设置 squashAnim1 动画的重复方式
    squashAnim1.setRepeatMode(ValueAnimator.REVERSE);
    // 设置 squashAnim1 动画的插值方式：减速插值
    squashAnim1.setInterpolator(new DecelerateInterpolator());
    // 定义小球“压扁”的动画：该动画控制小球的宽度加倍
    ValueAnimator squashAnim2 = ObjectAnimator.ofFloat(newBall,
        "width", newBall.getWidth()
        , newBall.getWidth() + BALL_SIZE);
    // 设置 squashAnim2 动画持续时间
    squashAnim2.setDuration(duration / 4);
    // 设置 squashAnim2 动画重复 1 次
    squashAnim2.setRepeatCount(1);
    // 设置 squashAnim2 动画的重复方式
    squashAnim2.setRepeatMode(ValueAnimator.REVERSE);
    // 设置 squashAnim2 动画的插值方式：减速插值
```

```
squashAnim2.setInterpolator(new DecelerateInterpolator());
// 定义小球“拉伸”的动画：该动画控制小球的 y 坐标“向下移”半个球
ObjectAnimator stretchAnim1 = ObjectAnimator.ofFloat(newBall
    , "y", endY, endY + BALL_SIZE / 2);
// 设置 stretchAnim1 动画持续时间
stretchAnim1.setDuration(duration / 4);
// 设置 stretchAnim1 动画重复 1 次
stretchAnim1.setRepeatCount(1);
// 设置 stretchAnim1 动画的重复方式
stretchAnim1.setRepeatMode(ValueAnimator.REVERSE);
// 设置 stretchAnim1 动画的插值方式：减速插值
stretchAnim1.setInterpolator(new DecelerateInterpolator());
// 定义小球“拉伸”的动画：该动画控制小球的高度减半
ValueAnimator stretchAnim2 = ObjectAnimator.ofFloat(newBall,
    "height", newBall.getHeight()
    , newBall.getHeight() - BALL_SIZE / 2);
// 设置 stretchAnim2 动画持续时间
stretchAnim2.setDuration(duration / 4);
// 设置 squashAnim2 动画重复 1 次
stretchAnim2.setRepeatCount(1);
// 设置 squashAnim2 动画的重复方式
stretchAnim2.setRepeatMode(ValueAnimator.REVERSE);
// 设置 squashAnim2 动画的插值方式：减速插值
stretchAnim2.setInterpolator(new DecelerateInterpolator());
// 定义小球“弹起”的动画
ObjectAnimator bounceBackAnim = ObjectAnimator.ofFloat(
    newBall , "y", endY, startY);
// 设置持续时间
bounceBackAnim.setDuration(duration);
// 设置动画的插值方式：减速插值
bounceBackAnim.setInterpolator(new DecelerateInterpolator());
// 使用 AnimatorSet 按顺序播放“掉落/压扁&拉伸/弹起动画
AnimatorSet bouncer = new AnimatorSet();
// 定义在 squashAnim1 动画之前播放 fallAnim 下落动画
bouncer.play(fallAnim).before(squashAnim1);
// 由于小球在“屏幕”下方弹起时，小球要被压扁
// 即：宽度加倍、x 坐标左移半个球，高度减半、y 坐标下移半个球
// 因此此处指定播放 squashAnim1 的同时
// 还播放 squashAnim2、stretchAnim1、stretchAnim2
bouncer.play(squashAnim1).with(squashAnim2);
bouncer.play(squashAnim1).with(stretchAnim1);
bouncer.play(squashAnim1).with(stretchAnim2);
// 指定播放 stretchAnim2 动画之后，播放 bounceBackAnim 弹起动画
bouncer.play(bounceBackAnim).after(stretchAnim2);
// 定义对 newBall 对象的 alpha 属性执行从 1 到 0 的动画（即定义渐隐动画）
ObjectAnimator fadeAnim = ObjectAnimator.ofFloat(newBall
    , "alpha", 1f, 0f);
// 设置动画持续时间
fadeAnim.setDuration(250);
// 为 fadeAnim 动画添加监听器
fadeAnim.addListener(new AnimatorListenerAdapter()
{
    // 当动画结束时
    @Override
    public void onAnimationEnd(Animator animation)
    {
        // 动画结束时将该动画关联的 ShapeHolder 删除
        balls.remove(((ObjectAnimator) animation).getTarget());
    }
});
// 再次定义一个 AnimatorSet 来组合动画
```

```
            AnimatorSet animatorSet = new AnimatorSet();
            // 指定在播放 fadeAnim 之前，先播放 bouncer 动画
            animatorSet.play(bouncer).before(fadeAnim);
            // 开发播放动画
            animatorSet.start();
            return true;
        }
        private ShapeHolder addBall(float x, float y)
        {
            // 创建一个圆
            OvalShape circle = new OvalShape();
            // 设置该椭圆的宽、高
            circle.resize(BALL_SIZE, BALL_SIZE);
            // 将圆包装成 Drawable 对象
            ShapeDrawable drawable = new ShapeDrawable(circle);
            // 创建一个 ShapeHolder 对象
            ShapeHolder shapeHolder = new ShapeHolder(drawable);
            // 设置 ShapeHolder 的 x、y 坐标
            shapeHolder.setX(x - BALL_SIZE / 2);
            shapeHolder.setY(y - BALL_SIZE / 2);
            int red = (int) (Math.random() * 255);
            int green = (int) (Math.random() * 255);
            int blue = (int) (Math.random() * 255);
            // 将 red、green、blue 三个随机数组合成 ARGB 颜色
            int color = 0xff000000 + red << 16 | green << 8 | blue;
            // 获取 drawable 上关联的画笔
            Paint paint = drawable.getPaint();
            // 将 red、green、blue 三个随机数除以 4 得到商值组合成 ARGB 颜色
            int darkColor = 0xff000000 | red / 4 << 16
                | green / 4 << 8 | blue / 4;
            // 创建圆形渐变
            RadialGradient gradient = new RadialGradient(
                37.5f, 12.5f, BALL_SIZE, color, darkColor
                , Shader.TileMode.CLAMP);
            paint.setShader(gradient);
            // 为 shapeHolder 设置 paint 画笔
            shapeHolder.setPaint(paint);
            balls.add(shapeHolder);
            return shapeHolder;
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            // 遍历 balls 集合中的每个 ShapeHolder 对象
            for (ShapeHolder shapeHolder : balls)
            {
                // 保存 canvas 的当前坐标系统
                canvas.save();
                // 坐标变换：将画布坐标系统平移到 shapeHolder 的 X、Y 坐标处
                canvas.translate(shapeHolder.getX()
                    , shapeHolder.getY());
                // 将 shapeHolder 持有的圆形绘制在 Canvas 上
                shapeHolder.getShape().draw(canvas);
                // 恢复 Canvas 坐标系统
                canvas.restore();
            }
        }
    }
}
```

上面的实例对小球使用了更多动画，因此小球掉落后，会再次弹起。

该实例的 ShapeHolder 需要对 width、height 增加动画，因此该 ShapeHolder 需要增加对 width、height 的 setter 和 getter 方法。下面是该实例中 ShapeHolder 的代码。

程序清单：codes\07\7.6\BouncingBalls\src\org\crazyit\image\ShapeHolder.java

```
public class ShapeHolder
{
    private float x = 0, y = 0;
    private ShapeDrawable shape;
    private int color;
    private RadialGradient gradient;
    private float alpha = 1f;
    private Paint paint;
    public ShapeHolder(ShapeDrawable s)
    {
        shape = s;
    }
    public float getWidth()
    {
        return shape.getShape().getWidth();
    }
    public void setWidth(float width)
    {
        Shape s = shape.getShape();
        s.resize(width, s.getHeight());
    }
    public float getHeight()
    {
        return shape.getShape().getHeight();
    }
    public void setHeight(float height)
    {
        Shape s = shape.getShape();
        s.resize(s.getWidth(), height);
    }
    // 省略其他属性的 setter 和 getter 方法
    ...
}
```

运行该实例，可以看到小球在底端压扁并弹起的动画，如图 7.19 所示。



图 7.18 小球下落

7.7 使用 SurfaceView 实现动画

虽然前面大量介绍了使用自定义 View 来进行绘图，但 View 的绘图机制存在如下缺陷：

- View 缺乏双缓冲机制。
- 当程序需要更新 View 上的图像时，程序必须重绘 View 上显示的整张图片。
- 新线程无法直接更新 View 组件。

由于 View 存在上述缺陷，所以通过自定义 View 来实现绘图，尤其是游戏中的绘图时性能并不好。Android 提供了一个 SurfaceView 来代替 View，在实现游戏绘图方面，SurfaceView 比 View 更加出色，因此一般推荐使用 SurfaceView。

7.7.1 SurfaceView 的绘图机制

SurfaceView 一般会与 SurfaceHolder 结合使用，SurfaceHolder 用于向与之关联的 SurfaceView 上绘图，调用 SurfaceView 的 getHolder()方法即可获取 SurfaceView 关联的 SurfaceHolder。

SurfaceHolder 提供了如下方法来获取 Canvas 对象。

- Canvas lockCanvas()：锁定整个 SurfaceView 对象，获取该 Surface 上的 Canvas。
- Canvas lockCanvas(Rect dirty)：锁定 SurfaceView 上 Rect 划分的区域，获取该 Surface 上的 Canvas。

当对同一个 SurfaceView 调用上面两个方法时，两个方法所返回的是同一个 Canvas 对象。但当程序调用第二个方法获取指定区域的 Canvas 时，SurfaceView 将只对 Rect 所“圈”出来的区域进行更新，通过这种方式可以提高画面的更新速度。

当通过 lockCanvas()获取指定了 SurfaceView 上的 Canvas 之后，接下来程序就可以调用 Canvas 进行绘图了，Canvas 绘图完成后通过如下方法来释放绘图、提交所绘制的图形：

- unlockCanvasAndPost(canvas);

需要指出的是，当调用 SurfaceHolder 的 unlockCanvasAndPost 方法之后，该方法之前所绘制的图形还处于缓冲之中，下一次 lockCanvas()方法锁定的区域可能会“遮挡”它。

下面的程序示范了 SurfaceView 的绘图机制。

程序清单：codes\07\7.7\SurfaceViewTest\src\org\crazyit\image\SurfaceViewTest.java

```
public class SurfaceViewTest extends Activity
{
    // SurfaceHolder 负责维护 SurfaceView 上绘制的内容
    private SurfaceHolder holder;
    private Paint paint;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        paint = new Paint();
        SurfaceView surface = (SurfaceView) findViewById(R.id.show);
        // 初始化 SurfaceHolder 对象
        holder = surface.getHolder();
        holder.addCallback(new Callback()
        {
            @Override
            public void surfaceChanged(SurfaceHolder arg0, int arg1, int arg2,
                    int arg3)
            {
            }
            @Override
            public void surfaceCreated(SurfaceHolder holder)
            {
                // 锁定整个 SurfaceView
                Canvas canvas = holder.lockCanvas();
                // 绘制背景
                Bitmap back = BitmapFactory.decodeResource(
                        SurfaceViewTest.this.getResources()
                        , R.drawable.sun);
                // 绘制背景
                canvas.drawBitmap(back, 0, 0, null);
                // 绘制完成，释放画布，提交修改
```

```
                holder.unlockCanvasAndPost(canvas);
                // 重新锁一次，"持久化"上次所绘制的内容
                holder.lockCanvas(new Rect(0, 0, 0, 0));
                holder.unlockCanvasAndPost(canvas);
            }
            @Override
            public void surfaceDestroyed(SurfaceHolder holder)
            {
            }
        });
        // 为 surface 的触摸事件绑定监听器
        surface.setOnTouchListener(new OnTouchListener()
        {
            @Override
            public boolean onTouch(View source, MotionEvent event)
            {
                // 只处理按下事件
                if (event.getAction() == MotionEvent.ACTION_DOWN)
                {
                    int cx = (int) event.getX();
                    int cy = (int) event.getY();
                    // 锁定 SurfaceView 的局部区域，只更新局部内容
                    Canvas canvas = holder.lockCanvas(new Rect(cx - 50,
                            cy - 50, cx + 50, cy + 50));
                    // 保存 canvas 的当前状态
                    canvas.save();
                    // 旋转画布
                    canvas.rotate(30, cx, cy);
                    paint.setColor(Color.RED);
                    // 绘制红色方块
                    canvas.drawRect(cx - 40, cy - 40, cx, cy, paint);
                    // 恢复 Canvas 之前的保存状态
                    canvas.restore();
                    paint.setColor(Color.GREEN);
                    // 绘制绿色方块
                    canvas.drawRect(cx, cy, cx + 40, cy + 40, paint);
                    // 绘制完成，释放画布，提交修改
                    holder.unlockCanvasAndPost(canvas);
                }
                return false;
            }
        });
    }
}
```

上面的程序还为 SurfaceHolder 添加了一个 Callback 实例，该 Callback 中定义了如下三个方法。

- void surfaceChanged(SurfaceHolder holder, int format, int width, int height)：当一个 surface 的格式或大小发生改变时回调该方法。
- void surfaceCreated(SurfaceHolder holder)：当 surface 被创建时回调该方法。
- void surfaceDestroyed(SurfaceHolder holder)：当 surface 将要被销毁时回调该方法。

上面的程序重写了 Callback 对象的 surfaceCreated()方法，并在该方法中为 SurfaceView 绘制了一个背景。为了避免背景图片被下一次 lockCanvas()遮挡，程序先调用了 holder.lockCanvas(new Rect(0, 0, 0, 0));，本次 lockCanvas 会“遮挡”上次 lockCanvas 所绘制

的图形，但由于本次 lockCanvas 的区域为 new Rect(0 ,0 , 0 ,0)，因此这里绘制的背景以后不会被遮挡了。

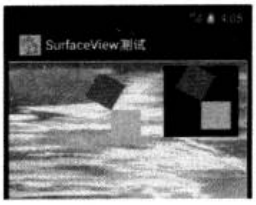


图 7.20　SurfaceView 的绘图机制

上面的程序监听了触摸屏事件，每次触碰屏幕时，程序会锁定触碰周围的区域（只更新该区域的数据），而且本次 lockCanvas()会遮挡上次 lockCanvas()后绘制的图形。运行上面的程序将看到如图 7.20 所示结果。

SurfaceView 上的“遮挡”有点类似于 Flash 上“蒙版”的概念，比如图 7.18 所示的第一次绘制的图形被第二次的 lockCanvas“遮挡”了；第三次 lockCanvas 时又可能“遮挡”第二次 lockCanvas 的区域，但不可能“遮挡”第一次 lockCanvas 的区域；如果第二次 lockCanvas“遮挡”的区域又被第三次 lockCanvas 所“遮挡”，那么原来第一次 drawCanvas 所绘制的图形可能“显露”出来。

实例：基于 SurfaceView 开发示波器

SurfaceView 与普通 View 还有一个重要的区别：View 的绘图必须在当前 UI 线程中进行——这也是前面程序需要更新 View 组件时总要采用 Handler 处理的原因；但 SurfaceView 就不会存在这个问题，因此 SurfaceView 的绘图是由 SurfaceHolder 来完成的。

对于 View 组件，如果程序需要花较长的时间来更新绘图，那么主 UI 线程将会被阻塞，无法响应用户的任何动作；而 SurfaceViewHolder 则会启用新的线程去更新 SurfaceView 的绘制，因此不会阻塞主 UI 线程。

一般来说，如果程序或游戏界面的动画元素较多，而且很多都需要通过定时器来控制这些动画元素的移动，就可以考虑使用 SurfaceView，而不是 View。例如下面我们使用 SurfaceView 开发一个示波器程序，该程序将会根据用户单击的按钮在屏幕上自动绘制正弦波或余弦波。

该程序的界面布局很简单，界面布局中包含两个按钮和一个 SurfaceView，程序每次绘制时只需要绘制（更新）当前点的波形，前面已经绘制的波形无须更新，这就利用了 SurfaceHolder 的 lockCanvas(Rect r)方法。

该程序的代码如下。

程序清单：codes\07\7.7\ShowWave\src\org\crazyit\image\ShowWave.java

```
public class ShowWave extends Activity
{
    private SurfaceHolder holder;
    private Paint paint;
    final int HEIGHT = 320;
    final int WIDTH = 320;
    final int X_OFFSET = 5;
    private int cx = X_OFFSET;
    //实际的 Y 轴的位置
    int centerY = HEIGHT / 2;
    Timer timer = new Timer();
    TimerTask task = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
final SurfaceView surface = (SurfaceView)
      findViewById(R.id.show);
// 初始化 SurfaceHolder 对象
holder = surface.getHolder();
paint = new Paint();
paint.setColor(Color.GREEN);
paint.setStrokeWidth(3);
Button sin = (Button)findViewById(R.id.sin);
Button cos = (Button)findViewById(R.id.cos);
OnClickListener listener = (new OnClickListener()
{
      @Override
      public void onClick(final View source)
      {
            drawBack(holder);
            cx = X_OFFSET;
            if(task != null)
            {
                  task.cancel();
            }
            task = new TimerTask()
            {
                  public void run()
                  {
                        int cy = source.getId() == R.id.sin ? centerY
                        - (int)(100 * Math.sin((cx - 5) * 2 * Math.PI
                        / 150))
                              : centerY - (int)(100 * Math.cos
                        ((cx - 5) * 2 * Math.PI / 150));
                        Canvas canvas = holder.lockCanvas(new Rect(cx , cy
                        - 2  , cx + 2
                        , cy + 2));
                        canvas.drawPoint(cx , cy , paint);
                        cx ++;
                        if (cx > WIDTH)
                        {
                        task.cancel();
                        task = null;
                        }
                        holder.unlockCanvasAndPost(canvas);
                  }
            };
            timer.schedule(task , 0 , 30);
      }
});
sin.setOnClickListener(listener);
cos.setOnClickListener(listener);
holder.addCallback(new Callback()
{
      @Override
      public void surfaceChanged(SurfaceHolder holder, int format,
            int width, int height)
      {
            drawBack(holder);
      }
      @Override
      public void surfaceCreated(final SurfaceHolder myHolder)
      {
      }
      @Override
      public void surfaceDestroyed(SurfaceHolder holder)
```

```
                {
                    timer.cancel();
                }
            });
        }
        private void drawBack(SurfaceHolder holder)
        {
            Canvas canvas = holder.lockCanvas();
            // 绘制白色背景
            canvas.drawColor(Color.WHITE);
            Paint p = new Paint();
            p.setColor(Color.BLACK);
            p.setStrokeWidth(2);
            // 绘制坐标轴
            canvas.drawLine(X_OFFSET , centerY , WIDTH , centerY , p);
            canvas.drawLine(X_OFFSET , 40 , X_OFFSET , HEIGHT , p);
            holder.unlockCanvasAndPost(canvas);
            holder.lockCanvas(new Rect(0 , 0 , 0 , 0));
            holder.unlockCanvasAndPost(canvas);
        }
    }
```

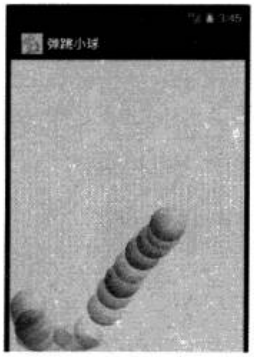


图 7.21　大珠小珠落玉盘

从上面的程序中两行粗体字代码可以看出，当程序每次绘制正弦波、余弦波上的当前点时，程序无须重绘整个画面，SurfaceHolder只要锁定当前绘制点的小范围即可，系统更新画面时也只要更新这个范围即可，因此具有较好的画面性能。运行该程序将看到结果如图 7.21 所示。

7.8　本章小结

本章主要介绍了 Android 的图形、图像处理，这种图形、图像处理不仅对 Android 界面开发十分重要，而且也是开发 Android 2D 游戏的基础。学习本章需要重点掌握 Android 丰富的绘图 API，包括 Canvas、Paint、Path 等类。除此之外，读者还需要掌握 Android 绘图的双缓冲机制、利用 Matrix 对图形进行几何变换等内容。除此之外，Android 提供的逐帧动画支持、补间动画、属性动画支持，尤其需要重点掌握属性动画。为了更好地开发游戏动画界面，Android 专门提供了 SurfaceView，本章详细讲解了 SurfaceView 的绘图机制，并讲解了如何继承 SurfaceView 来开发动画。

CHAPTER

8

第8章 Android 数据存储与 IO

本章要点

- SharedPreferences 的概念和作用
- 使用 SharedPreferences 保存程序的参数、选项
- 读写、其他应用的 SharedPreferences
- Android 的文件 IO
- 读、写 SD 卡上的文件
- 了解 SQLite 数据库
- 使用 Android 的 API 操作 SQLite 数据库
- 使用 sqlite3 工具管理 SQLite 数据库
- SQLiteOpenHelper 类的功能和用法
- Android 的手势支持
- 手势检测
- 向手势库中添加手势
- 识别用户手势
- 自动朗读支持

所有应用程序都必然涉及数据的输入、输出，Android 应用也不例外，应用程序的参数设置、程序运行状态数据这些都需要保存到外部存储器上，这样系统关机之后数据才不会丢失。Android 应用开发是使用 Java 语言来开发的，因此开发者在 Java IO 中的编程经验大部分都可“移植”到 Android 应用开发上，但 Android 系统还提供了一些专门的 IO API，通过这些 API 可以更有效地进行输入、输出。

如果应用程序只有少量数据需要保存，那么使用普通文件就可以了；但如果应用程序有大量数据需要存储、访问，就需要借助于数据库了，Android 系统内置了 SQLite 数据库，SQLite 数据库是一个真正轻量级的数据库，它没有后台进程，整个数据库就对应于一个文件，这样可以非常方便地在不同设备之间移植。Android 不仅内置了 SQLite 数据库，而且为访问 SQLite 数据库提供了大量便捷的 API。本章将会详细介绍如何在 Android 应用中使用 SQLite 数据库。

8.1 使用 SharedPreferences

有些时候，应用程序有少量的数据需要保存，而且这些数据的格式很简单：都是普通的字符串、标量类型的值等，比如应用程序的各种配置信息（如是否打开音效、是否使用振动效果等）、小游戏的玩家积分（如扫雷英雄榜之类的）等，对于这种数据，Android 提供了 SharedPreferences 进行保存。

8.1.1 SharedPreferences 与 Editor 简介

SharedPreferences 保存的数据主要是类似于配置信息格式的数据，因此它保存的数据主要是简单类型的 key-value 对。

SharedPreferences 接口主要负责读取应用程序的 Preferences 数据，它提供了如下常用方法来访问 SharedPreferences 中的 key-value 对。

- boolean contains(String key)：判断 SharedPreferences 是否包含特定 key 的数据。
- abstract Map<String, ?> getAll()：获取 SharedPreferences 数据里全部的 key-value 对。
- boolean getXxx(String key, xxx defValue)：获取 SharedPreferences 数据里指定 key 对应的 value。如果该 key 不存在，返回默认值 defValue。其中 xxx 可以是 boolean、float、int、long、String 等各种基本类型的值。

SharedPreferences 接口本身并没有提供写入数据的能力，而是通过 SharedPreferences 的内部接口，SharedPreferences 调用 edit()方法即可获取它所对应的 Editor 对象。Editor 提供了如下方法来向 SharedPreferences 写入数据。

- SharedPreferences.Editor clear()：清空 SharedPreferences 里所有数据。
- SharedPreferences.Editor putXxx(String key, xxx value)：向 SharedPreferences 存入指定 key 对应的数据。其中 xxx 可以是 boolean、float、int、long、String 等各种基本类型的值。
- SharedPreferences.Editor remove(String key)：删除 SharedPreferences 里指定 key 对应的数据项。
- boolean commit()：当 Editor 编辑完成后，调用该方法提交修改。



提示：
从用法角度来看，SharedPreferences 和 SharedPreferences.Editor 组合起来非常像 Map，其中 SharedPreferences 负责根据 key 读取数据，而 SharedPreferences.Editor 则用于写入数据。

SharedPreferences 本身是一个接口，程序无法直接创建 SharedPreferences 实例，只能通过 Context 提供的 getSharedPreferences(String name , int mode)方法来获取 SharedPreferences 实例，该方法的第二个参数支持如下几个值。

- Context.MODE_PRIVATE：指定该 SharedPreferences 数据只能被本应用程序读、写。
- Context. MODE_WORLD_READABLE：指定该 SharedPreferences 数据能被其他应用程序读，但不能写。
- Context. MODE_WORLD_WRITEABLE：指定该 SharedPreferences 数据能被其他应用程序读、写。

下面介绍对 SharedPreferences 的简单读、写。

8.1.2 SharedPreferences 的存储位置和格式

下面的程序示范了如何向 SharedPreferences 中写入、读取数据，该程序的界面很简单，它只是提供了两个按钮，其中一个用于写入数据，另外一个用于读取数据，故此处不再给出界面布局文件。程序代码如下。

程序清单：codes\08\8.1\SharedPreferencesTest\src\org\crazyit\io\SharedPreferencesTest.java

```
public class SharedPreferencesTest extends Activity
{
    SharedPreferences preferences;
    SharedPreferences.Editor editor;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取只能被本应用程序读、写的 SharedPreferences 对象
        preferences = getSharedPreferences("crazyit", MODE_WORLD_READABLE);
        editor = preferences.edit();
        Button read = (Button) findViewById(R.id.read);
        Button write = (Button) findViewById(R.id.write);
        read.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 读取字符串数据
                String time = preferences.getString("time", null);
                // 读取 int 类型的数据
                int randNum = preferences.getInt("random", 0);
                String result = time == null ? "您暂时还未写入数据" : "写入时间为："
                    + time + "\n 上次生成的随机数为：" + randNum;
                // 使用 Toast 提示信息
                Toast.makeText(SharedPreferencesTest.this, result, 5000).show();
            }
```

```
        });
        write.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                SimpleDateFormat sdf = new SimpleDateFormat("yyyy年MM月dd日 "
                        + "hh:mm:ss");
                // 存入当前时间
                editor.putString("time", sdf.format(new Date()));
                // 存入一个随机数
                editor.putInt("random", (int) (Math.random() * 100));
                // 提交所有存入的数据
                editor.commit();
            }
        });
    }
}
```

上面的程序中第一段粗体字代码用于读取 SharedPreferences 数据，当程序所读取的 SharedPreferences 文件根本不存在时，程序也返回默认值，并不会抛出异常；程序的第二段粗体字代码用于写入 SharedPreferences 数据，由于 SharedPreferences 并不支持写入 Date 类型的值，故程序使用了 SimpleDateFormat 将 Date 格式化成字符串后写入。

运行上面的程序，单击程序中“写入数据”按钮，程序将完成 SharedPreferences 写入，写入完成后打开 DDMS 的 File Explorer 面板（参考第 1 章介绍的方法来打开 DDMS 的 File Explorer 面板），然后展开文件浏览树，看到如图 8.1 所示的窗口。



图 8.1　SharedPreferences 的存储路径

从图 8.1 可以看出，SharedPreferences 数据总是保存在/data/data/<package name>/shared_prefs 目录下，SharedPreferences 数据总是以 XML 格式保存。通过 File Explorer 面板的导出文件按钮将该 XML 文件导出到 XML 文档，打开该 XML 文档可看到如下文件内容：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<int name="random" value="86" />
<string name="time">2012年11月08日 04:39:39</string>
</map>
```

从上面的文件不难看出，SharedPreferences 数据文件是一个根元素为<map.../>的根元素，该元素里每个子元素代表一个 key-value 对，当 value 是整数类型的值时使用<int.../>子元素，当 value 是字符串类型时，使用<string .../>子元素……依此类推。



图 8.2　读取 SharedPreferences 数据

单击上面程序中的“读取数据”按钮，程序弹出一个 Toast 对话框显示上次写入的数据，如图 8.2 所示。

实例：记录应用程序的使用次数

这个简单的实例可以记住应用程序的使用次数：当用户第一次启动该应用程序时，系统创建 SharedPreferences 来记录使用次数。用户以后启动应用程序时，系统先读取 SharedPreferences 中记录的使用次数，然后将使用次数加 1。

本实例程序的代码如下。

程序清单：codes\08\8.1\UseCount\src\org\crazyit\io\UseCount.java

```
public class UseCount extends Activity
{
    SharedPreferences preferences;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        preferences = getSharedPreferences("count"
            , MODE_WORLD_READABLE);
        // 读取 SharedPreferences 里的 count 数据
        int count = preferences.getInt("count", 0);
        // 显示程序以前使用的次数
        Toast.makeText(this, "程序以前被使用了" + count + "次。"
            , Toast.LENGTH_LONG).show();
        Editor editor = preferences.edit();
        // 存入数据
        editor.putInt("count", ++count);
        // 提交修改
        editor.commit();
    }
}
```

上面的程序中第一行粗体字代码用于读取 SharedPreferences 中记录的使用次数；第二行粗体字代码将使用次数增加 1，并再次将使用次数写入 SharedPreferences 中。

8.1.3 读、写其他应用 SharedPreferences

要读、写其他应用的 SharedPreferences，前提是创建该 SharedPreferences 的应用程序指定相应的访问权限，例如指定了 MODE_WORLD_READABLE，这表明该 SharedPreferences 可被其他应用程序读取；指定了 MODE_WORLD_WRITABLE，这表明该 SharedPreferences 可被其他程序写入。

例如上一个示例创建 SharedPreferences 时就指定了 MODE_WORLD_READABLE 模式，这表明该 SharedPreferences 数据可以被其他程序读取。

为了读取其他程序的 SharedPreferences，可按如下步骤进行。

（1）需要创建其他程序对应的 Context，例如如下代码：

```
useCount = createPackageContext("org.crazyit.io"
    , Context.CONTEXT_IGNORE_SECURITY);
```

上面的程序中 org.crazyit.io 就是其他程序的包名——实际上 Android 系统就是用应用程序的包名来作为该程序的标识的。



提示：

访问其他应用程序的 SharedPreferences 的关键就是获取其他应用程序的 Context。前面已经指出，Context 代表了访问该 Android 应用的全局信息的接口，而 Android 应用的包名正是该应用的唯一标识，因此程序可根据 Android 应用的包名来获取相应的 Context。

（2）调用其他应用程序的 Context 的 getSharedPreferences(String name , int mode)即可获取相应的 SharedPreferences 对象。

（3）如果需要向其他应用的 SharedPreferences 数据写入数据，调用 SharedPreferences 的 edit()方法获取相应的 Editor 即可。

下面的程序示范如何读取上一个程序所保存的 SharedPreferences 数据。

程序清单：codes\08\8.1\ReadOtherPreferences\src\org\crazyit\other\ReadOtherPreferences.java

```
public class ReadOtherPreferences extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Context useCount = null;
        try
        {
            // 获取其他程序所对应的 Context
            useCount = createPackageContext("org.crazyit.io",
                Context.CONTEXT_IGNORE_SECURITY);
        }
        catch (NameNotFoundException e)
        {
            e.printStackTrace();
        }
        // 使用其他程序的 Context 获取对应的 SharedPreferences
        SharedPreferences prefs = useCount.getSharedPreferences("count",
            Context.MODE_WORLD_READABLE);
        // 读取数据
        int count = prefs.getInt("count", 0);
        TextView show = (TextView) findViewById(R.id.show);
        // 显示读取的数据内容
        show.setText("UseCount 应用程序以前被使用了" + count + "次。");
    }
}
```

上面的程序中两行粗体字代码就是读取其他应用程序的 SharedPreferences 的关键代码。

事实上，如果开发者不通过先获取其他应用程序的 Context，再获取 SharedPreferences 的方式也可读取 SharedPreferences 的数据——开发者完全使用以 IO 流的方式先读取 SharedPreferences 对应的 XML 文件，再通过 XML 解析来获取数据也是可行的，只是这种方式过于烦琐，而使用 SharedPreferences 来读写数据则简单得多。

8.2 File 存储

读者学习 Java SE 的时候都知道 Java 提供了一套完整的 I/O 流体系，包括 FileInputStream、

FileOutputStream 等，通过这些 I/O 流可以非常方便地访问磁盘上的文件内容。Android 同样支持以这种方式来访问手机存储器上的文件。

8.2.1 openFileOutput 和 openFileInput

Context 提供了如下两个方法来打开本应用程序的数据文件夹里的文件 I/O 流。

- FileInputStream openFileInput(String name)：打开应用程序的数据文件夹下的 name 文件对应输入流。
- FileOutputStream openFileOutput(String name, int mode)：：打开应用程序的数据文件夹下的 name 文件对应输出流。

上面两个方法分别用于打开文件输入流、输出流，其中第二个方法的第二个参数指定打开文件的模式，该模式支持如下值。

- MODE_PRIVATE：该文件只能被当前程序读写。
- MODE_APPEND：以追加方式打开该文件，应用程序可以向该文件中追加内容。
- MODE_WORLD_READABLE：该文件的内容可以被其他程序读取。
- MODE_WORLD_WRITEABLE：该文件的内容可由其他程序读、写。

除此之外，Context 还提供了如下几个方法来访问应用程序的数据文件夹。

- getDir(String name, int mode)：在应用程序的数据文件夹下获取或创建 name 对应的子目录。
- File getFilesDir()：获取该应用程序的数据文件夹的绝对路径。
- String[] fileList()：返回该应用程序的数据文件夹下的全部文件。
- deleteFile(String)：删除该应用程序的数据文件夹下的指定文件。

下面的程序简单示范了如何读写应用程序数据文件夹内的文件。该程序的界面布局同样很简单，只包含两个文本框和两个按钮；其中第一组文本框和按钮用于处理写入，文本框用于接受用户输入，当用户按下“写入”按钮时，程序将会把数据写入文件；第二组文本框和按钮用于处理读取，当用户按下“读取”按钮时，该文本框显示文件中的数据。程序代码如下。

注意：

由于本书主要介绍 Android 应用开发的相关知识，因此 Java IO 支持的相关内容则不在本书介绍的范围之内，如果读者对 Java IO 还不熟悉，建议先阅读疯狂 Java 体系的《疯狂 Java 讲义》。



程序清单：codes\08\8.2\FileTest\src\org\crazyit\io\FileTest.java

```
public class FileTest extends Activity
{
    final String FILE_NAME = "crazyit.bin";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        System.out.println(new StringBuilder("a").append("b").append("c")
            .toString());
```

```
        // 获取两个按钮
        Button read = (Button) findViewById(R.id.read);
        Button write = (Button) findViewById(R.id.write);
        // 获取两个文本框
        final EditText edit1 = (EditText) findViewById(R.id.edit1);
        final EditText edit2 = (EditText) findViewById(R.id.edit2);
        // 为 write 按钮绑定事件监听器
        write.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 将 edit1 中的内容写入文件中
                write(edit1.getText().toString());
                edit1.setText("");
            }
        });
        read.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 读取指定文件中的内容，并显示出来
                edit2.setText(read());
            }
        });
    }
    private String read()
    {
        try
        {
            // 打开文件输入流
            FileInputStream fis = openFileInput(FILE_NAME);
            byte[] buff = new byte[1024];
            int hasRead = 0;
            StringBuilder sb = new StringBuilder("");
            // 读取文件内容
            while ((hasRead = fis.read(buff)) > 0)
            {
                sb.append(new String(buff, 0, hasRead));
            }
            // 关闭文件输入流
            fis.close();
            return sb.toString();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return null;
    }
    private void write(String content)
    {
        try
        {
            // 以追加模式打开文件输出流
            FileOutputStream fos = openFileOutput(FILE_NAME, MODE_APPEND);
            // 将 FileOutputStream 包装成 PrintStream
            PrintStream ps = new PrintStream(fos);
            // 输出文件内容
            ps.println(content);
```

```
                // 关闭文件输出流
                ps.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

上面的程序中第一段粗体字代码用于读取应用程序的数据文件，第二段粗体字代码用于向应用程序的数据文件中追加内容。从上面的粗体字代码可以看出，当 Android 系统调用 Context 的 openFileInput()、openFileOutput()打开文件输入流或输出流之后，接下来 I/O 流的用法与 Java SE 中 I/O 流的用法完全一样：想直接用节点流读写也行，用包装流包装之后再处理也没问题。

当按下程序中的“写入”按钮时，用户在第一个文本框中输入的内容将会被保存到应用程序的数据文件中，打开 File Explorer 面板后，可以看到如图 8.3 所示的界面。

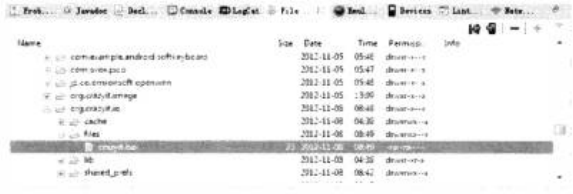


图 8.3 应用程序的数据文件

从图 8.3 可以看出，应用程序的数据文件默认保存在/data/data/<package name>/files 目录下。

8.2.2 读写 SD 卡上的文件

当程序通过 Context 的 openFileInput 或 openFileOutput 来打开文件输入流、输出流时，程序所打开的都是应用程序的数据文件夹里的文件，这样所存储的文件大小可能比较有限——毕竟手机内置的存储空间是有限的。

为了更好地存、取应用程序的大文件数据，应用程序需要读、写 SD 卡上的文件。SD 卡大大扩充手机的存储能力。

读、写 SD 上的文件请按如下步骤进行。

（1）调用 Environment 的 getExternalStorageState()方法判断手机上是否插入了 SD 卡，并且应用程序具有读写 SD 卡的权限。例如使用如下代码：

```
// 如果手机已插入 SD 卡，且应用程序具有读写 SD 卡的能力，下面语句返回 true
Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED)
```

（2）调用 Environment 的 getExternalStorageDirectory()方法来获取外部存储器，也就是 SD 卡的目录。

（3）使用 FileInputStream、FileOutputStream、FileReader 或 FileWriter 读、写 SD 卡里的文件。

应用程序读、写 SD 卡的文件有如下两个注意点：

- 手机上应该已插入 SD 卡。对于模拟器来说，可通过 mksdcard 命令来创建虚拟存

储卡，关于虚拟存储卡的管理参考第 1 章。

- 为了读、写 SD 卡上的数据，必须在应用程序的清单文件（AndroidManifest.xml）中添加读、写 SD 卡的权限。例如如下配置：

```
<!-- 在 SD 卡中创建与删除文件权限 -->
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 向 SD 卡写入数据权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

下面的程序示范了如何读、写 SD 卡上的文件，该程序的主界面与上一个程序的界面完全相同。支持该程序数据读、写是基于 SD 卡的。该程序代码如下。

程序清单：codes\08\8.2\SDCardTest\src\org\crazyit\io\SDCardTest.java

```
public class SDCardTest extends Activity
{
    final String FILE_NAME = "/crazyit.bin";
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取两个按钮
        Button read = (Button) findViewById(R.id.read);
        Button write = (Button) findViewById(R.id.write);
        // 获取两个文本框
        final EditText edit1 = (EditText) findViewById(R.id.edit1);
        final EditText edit2 = (EditText) findViewById(R.id.edit2);
        // 为 write 按钮绑定事件监听器
        write.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 将 edit1 中的内容写入文件中
                write(edit1.getText().toString());
                edit1.setText("");
            }
        });
        read.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 读取指定文件中的内容，并显示出来
                edit2.setText(read());
            }
        });
    }
    private String read()
    {
        try
        {
            // 如果手机插入了 SD 卡，而且应用程序具有访问 SD 的权限
            if (Environment.getExternalStorageState().equals(
                    Environment.MEDIA_MOUNTED))
            {
                // 获取 SD 卡对应的存储目录
                File sdCardDir = Environment.getExternalStorageDirectory();
                // 获取指定文件对应的输入流
                FileInputStream fis = new FileInputStream(
```

```
                    sdCardDir.getCanonicalPath() + FILE_NAME);
                // 将指定输入流包装成 BufferedReader
                BufferedReader br = new BufferedReader(new
                    InputStreamReader(fis));
                StringBuilder sb = new StringBuilder("");
                String line = null;
                // 循环读取文件内容
                while ((line = br.readLine()) != null)
                {
                    sb.append(line);
                }
                // 关闭资源
                br.close();
                return sb.toString();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return null;
    }
    private void write(String content)
    {
        try
        {
            // 如果手机插入了 SD 卡，而且应用程序具有访问 SD 的权限
            if (Environment.getExternalStorageState().equals(
                Environment.MEDIA_MOUNTED))
            {
                // 获取 SD 卡的目录
                File sdCardDir = Environment.getExternalStorageDirectory();
                File targetFile = new File(sdCardDir
                    .getCanonicalPath() + FILE_NAME);
                // 以指定文件创建 RandomAccessFile 对象
                RandomAccessFile raf = new RandomAccessFile(
                    targetFile, "rw");
                // 将文件记录指针移动到最后
                raf.seek(targetFile.length());
                // 输出文件内容
                raf.write(content.getBytes());
                // 关闭 RandomAccessFile
                raf.close();
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

上面的程序中第一段粗体字代码用于读取 SD 卡中指定文件的内容；第二段粗体字代码则使用 RandomAccessFile 向 SD 卡指定文件追加内容——如果使用 FileOutputStream 向指定文件写入数据，FileOutputStream 会把原有的文件内容清空，那就不是追加文件内容了。由此可见，当程序直接使用 FileOutputStream 进行输出时，比如使用 Context 的 openFileOutput 方便。

运行上面的程序，在第一个文本框内输入一些字符串，然后单击“写入”按钮即可将数

据写入底层 SD 卡。打开 File Explorer，即可看到如图 8.4 所示的界面。

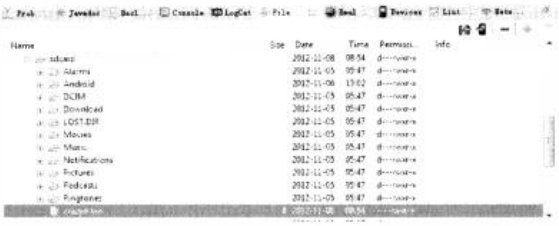


图 8.4　存入 SD 卡的数据

需要指出的是，当开发者直接在 Eclipse 中运行 Android 应用程序时，Eclipse 默认启动的模拟器是不带 SD 卡的，为了让 Eclipse 启动的模拟器带上 SD 卡，可以通过 Run As→Run Configurations 菜单项打开如图 8.5 所示的对话框，通过该对话框下面的附加选项来使用 SD 卡（实际上就是在启动模拟器时增加-sdcard 选项）。

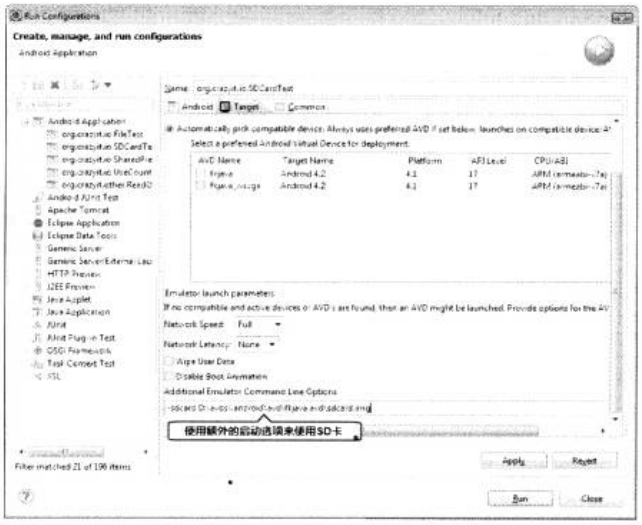


图 8.5　指定启动模拟器时使用 SD 卡



提示：

如果开发者不想使用 Environment.getExternalStorageDirectory()这么复杂的语句来获取 SD 卡的路径，完全可以使用/mnt/sdcard/路径代表 SD 卡的路径，然后程序通过判断/mnt/sdcard/路径是否存在就可知道手机是否已插入了 SD 卡。

实例：SD 卡文件浏览器

下面我们将会利用 Java 的 File 类开发一个 SD 卡文件浏览器，该程序直接使用/mnt/sdcard 来访问系统的 SD 卡目录，然后通过 File 的 listFile()方法来获取指定目录下的全部文件和文件夹。

当程序启动时，系统启动获取/mnt/sdcard 目录下的全部文件、文件夹，并使用 ListView 将它们显示出来；当用户单击 ListView 的指定列表项时，系统将会显示该列表项下全部文件夹和文件。

该程序的界面布局文件如下。

程序清单：codes\08\8.2\SDFileExplorer\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      >
<!-- 显示当前路径的的文本框 -->
<TextView
      android:id="@+id/path"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal"
      android:layout_alignParentTop="true"
      />
<!-- 列出当前路径下所有文件的 ListView -->
<ListView
      android:id="@+id/list"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:divider="#000"
      android:dividerHeight="1px"
      android:layout_below="@id/path"
      />
<!-- 返回上一级目录的按钮 -->
<Button android:id="@+id/parent"
      android:layout_width="38dp"
      android:layout_height="34dp"
      android:background="@drawable/home"
      android:layout_centerHorizontal="true"
      android:layout_alignParentBottom="true"
/>
</RelativeLayout>
```

该程序主要利用了 File 的 listFile 来列出指定目录的全部文件，程序代码如下。

程序清单：codes\08\8.2\SDFileExplorer\src\org\crazyit\io\SDFileExplorer.java

```
public class SDFileExplorer extends Activity
{
      ListView listView;
      TextView textView;
      // 记录当前的父文件夹
      File currentParent;
      // 记录当前路径下的所有文件的文件数组
      File[] currentFiles;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取列出全部文件的 ListView
            listView = (ListView) findViewById(R.id.list);
            textView = (TextView) findViewById(R.id.path);
            // 获取系统的 SD 卡的目录
            File root = new File("/mnt/sdcard/");
            // 如果 SD 卡存在
            if (root.exists())
            {
                  currentParent = root;
                  currentFiles = root.listFiles();
                  // 使用当前目录下的全部文件、文件夹来填充 ListView
```

```
            inflateListView(currentFiles);
        }
        // 为ListView的列表项的单击事件绑定监听器
        listView.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                // 用户单击了文件，直接返回，不做任何处理
                if (currentFiles[position].isFile()) return;
                // 获取用户点击的文件夹下的所有文件
                File[] tmp = currentFiles[position].listFiles();
                if (tmp == null || tmp.length == 0)
                {
                    Toast.makeText(SDFileExplorer.this
                        , "当前路径不可访问或该路径下没有文件",
                        Toast.LENGTH_SHORT).show();
                }
                else
                {
                    // 获取用户单击的列表项对应的文件夹，设为当前的父文件夹
                    currentParent = currentFiles[position]; //②
                    // 保存当前的父文件夹内的全部文件和文件夹
                    currentFiles = tmp;
                    // 再次更新ListView
                    inflateListView(currentFiles);
                }
            }
        });
        // 获取上一级目录的按钮
        Button parent = (Button) findViewById(R.id.parent);
        parent.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                try
                {
                    if (!currentParent.getCanonicalPath()
                        .equals("/mnt/sdcard"))
                    {
                        // 获取上一级目录
                        currentParent = currentParent.getParentFile();
                        // 列出当前目录下所有文件
                        currentFiles = currentParent.listFiles();
                        // 再次更新ListView
                        inflateListView(currentFiles);
                    }
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        });
    }
    private void inflateListView(File[] files)  //①
    {
        // 创建一个List集合，List集合的元素是Map
        List<Map<String, Object>> listItems =
```

```
            new ArrayList<Map<String, Object>>();
        for (int i = 0; i < files.length; i++)
        {
            Map<String, Object> listItem =
                new HashMap<String, Object>();
            // 如果当前 File 是文件夹，使用 folder 图标；否则使用 file 图标
            if (files[i].isDirectory())
            {
                listItem.put("icon", R.drawable.folder);
            }
            else
            {
                listItem.put("icon", R.drawable.file);
            }
            listItem.put("fileName", files[i].getName());
            // 添加 List 项
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(this
            , listItems, R.layout.line
            , new String[]{ "icon", "fileName" }
            , new int[]{R.id.icon, R.id.file_name });
        // 为 ListView 设置 Adapter
        listView.setAdapter(simpleAdapter);
        try
        {
            textView.setText("当前路径为："
                + currentParent.getCanonicalPath());
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

上面的程序中①号粗体字方法使用 File[]数组来填充 ListView，填充时程序会根据 File[]数组里的数据元素（File 对象）代表的是文件还是文件夹来选择使用文件图标或是文件夹图标，如该方法中粗体字代码所示。

当用户单击 ListView 中某个列表项时，上面的程序会将用户单击的列表项所对应的文件夹当成 currentParent 处理，并再次调用 inflateListView(File[] files)方法来列出当前文件夹下的所有文件。

运行上面的程序，将看到如图 8.6 所示的界面。

正如图 8.6 中所示，该程序就像 SD 卡资源管理器一样可以非常方便地浏览 SD 卡里包含的全部文件。



图 8.6 SD 卡文件浏览器



提示：

从图 8.6 所示界面中可看到/mnt/sdcard/abc 目录下包含了 haha、hehe 两个目录和 color.gif 文件。这就要求开发者的模拟器里的 SD 卡里带有这些目录和文件，为了在 SD 卡里创建目录，可先运行 adb shell 命令来启动 Android 的 Shell 窗口——由于 Android 的内核就是 Linux，因此可在该 Shell 窗口里执行 ls、mkdir 等常见的 Linux 命令。

8.3 SQLite 数据库

Android 系统集成了一个轻量级的数据库：SQLite，SQLite 并不想成为像 Oracle、MySQL 那样的数据库。SQLite 只是一个嵌入式的数据库引擎，专门适用于资源有限的设备上（如手机、PDA 等）适量数据存取。

虽然 SQLite 支持绝大部分 SQL 92 语法，也允许开发者使用 SQL 语句操作数据库中的数据，但 SQLite 并不像 Oracle、MySQL 数据库那样需要安装、启动服务器进程，SQLite 数据库只是一个文件。



提示：

从本质上来看，SQLite 的操作方式只是一种更为便捷的文件操作。后面我们会看到，当应用程序创建或打开一个 SQLite 数据库时，其实只是打开一个文件准备读写，因此有人说 SQLite 有点像 Microsoft 的 Access（实际上 SQLite 功能要强大得多）。可能有读者需要问，如果实际项目中有大量数据需要读写，而且需要面临大量用户的并发存储怎么办呢？对于这种情况，本身就不应该把数据存放在手机的 SQLite 数据库里——毕竟手机还是手机，它的存储能力、计算能力都不足以让它充当服务器的角色。

8.3.1 SQLiteDatabase 简介

Android 提供了 SQLiteDatabase 代表一个数据库（底层就是一个数据库文件），一旦应用程序获得了代表指定数据库的 SQLiteDatabase 对象，接下来就可通过 SQLiteDatabase 对象来管理、操作数据库了。

SQLiteDatabase 提供了如下静态方法来打开一个文件对应的数据库。

- static SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)：打开 path 文件所代表的 SQLite 数据库。
- static SQLiteDatabase openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)：打开或创建（如果不存在）file 文件所代表的 SQLite 数据库。
- static SQLiteDatabase openOrCreateDatabase(String path,SQLiteDatabase.CursorFactory factory)：打开或创建（如果不存在）path 文件所代表的 SQLite 数据库。

在程序中获取 SQLiteDatabase 对象之后，接下来就可调用 SQLiteDatabase 的如下方法来操作数据库了。

- execSQL(String sql, Object[] bindArgs)：执行带占位符的 SQL 语句。
- execSQL(String sql)：执行 SQL 语句。
- insert(String table, String nullColumnHack, ContentValues values)：向执行表中插入数据。
- update(String table, ContentValues values, String whereClause, String[] whereArgs)：更新指定表中的特定数据。
- delete(String table, String whereClause, String[] whereArgs)：删除指定表中的特定数据。

- Cursor query(String table, String[] columns, String whereClause, String[] whereArgs, String groupBy, String having, String orderBy)：对执行数据表执行查询。
- Cursor query(String table, String[] columns, String whereClause, String[] whereArgs, String groupBy, String having, String orderBy, String limit)：对执行数据表执行查询。limit 参数控制最多查询几条记录（用于控制分页的参数）。
- Cursor query(boolean distinct, String table, String[] columns, String whereClause, String[]whereArgs, String groupBy, String having, String orderBy, String limit)：对指定表执行查询语句。其中第一个参数控制是否去除重复值。
- rawQuery(String sql, String[] selectionArgs)：执行带占位符的 SQL 查询。
- beginTransaction()：开始事务。
- endTransaction()：结束事务。

从上面的方法不难看出，其实 SQLiteDatabase 的作用有点类似于 JDBC 的 Connection 接口，但 SQLiteDatabase 提供的方法更多：比如 insert、update、delete、query 等方法，其实这些方法完全可通过执行 SQL 语句来完成，但 Android 考虑到部分开发者对 SQL 语法不熟悉，所以提供这些方法帮助开发者以更简单的方式来操作数据表的数据。

上面的查询方法都是返回一个 Cursor 对象，Android 中的 Cursor 类似于 JDBC 的 ResultSet，Cursor 同样提供了如下方法来移动查询结果的记录指针。

- move(int offset)：将记录指针向上或向下移动指定的行数。offset 为正数就是向下移动；为负数就是向上移动。
- boolean moveToFirst()：将记录指针移动到第一行，如果移动成功则返回 true。
- boolean moveToLast()：将记录指针移动到最后一行，如果移动成功则返回 true。
- boolean moveToNext()：将记录指针移动到下一行，如果移动成功则返回 true。
- boolean moveToPosition(int position)：将记录指针移动到指定的行，如果移动成功则返回 true。
- boolean moveToPrevious()：将记录指针移动到上一行，如果移动成功则返回 true。

一旦将记录指针移动到指定行之后，接下来就可以调用 Cursor 的 getXxx()方法获取该行的指定列的数据。



提示：

其实如果读者具有 JDBC 编程的经验，完全可以把 SQLiteDatabase 当成 JDBC 中 Connection 和 Statement 的混合体——因为 SQLiteDatabase 既代表了与数据库的连接，也可直接用于执行 SQL 操作；而 Android 中 Cursor 则可当成 ResultSet，而且 Cursor 提供了更多便捷的方法来操作结果集。实际上对于一个 Java 程序员来说，JDBC 几乎是必备的编程基础，如果读者需要详细学习 JDBC 的相关知识，可以参考疯狂 Java 体系的《疯狂 Java 讲义》一书。

8.3.2 创建数据库和表

前面已经讲到，使用 SQLiteDatabase 的静态方法即可打开或创建数据库，例如如下代码：

```
SQLiteDatabase.openOrCreateDatabase("/mnt/db/temp.db3",null);
```

上面的代码就用于打开或创建一个 SQLite 数据库，如果/mnt/db/目录下的 temp.db3 文件

（该文件就是一个数据库）存在，那么程序就是打开该数据库；如果该文件不存在，则上面的代码将会在该目录下创建 temp.db3 文件（即对应于数据库）。

上面的代码中没有指定 SQLiteDatabase.CursorFactory 参数，该参数是一个用于返回 Cursor 的工厂，如果指定该参数为 null，则意味着使用默认的工厂。

上面的代码即可返回一个 SQLiteDatabase 对象，该对象的 execSQL 可执行任意的 SQL 语句，因此程序可通过如下代码在程序中创建数据表：

```
// 定义建表语句
sql = "create table user_inf(user_id integer primary key ,"
      + " user_name varchar(255),"
      + " user_pass varchar(255))"
// 执行 SQL 语句
db.execSQL(sql);
```

在程序中执行上面的代码即可在数据库中创建一个数据表。

8.3.3 使用 SQL 语句操作 SQLite 数据库

正如前面提到的，SQLiteDatabase 的 execSQL 方法可执行任意 SQL 语句，包括带占位符的 SQL 语句。但由于该方法没有返回值，一般用于执行 DDL 语句或 DML 语句；如果需要执行查询语句，则可调用 SQLiteDatabase 的 rawQuery(String sql, String[] selectionArgs)方法。

例如如下代码可用于执行 DML 语句，

```
// 执行插入语句
db.execSQL("insert into news_inf values(null , ? , ?)"
      , new String[]{title , content});
```

下面的程序示范了如何在 Android 应用中操作 SQLite 数据库，该程序提供了两个文本框，用户可以在这两个文本框中输入内容，当用户单击“插入”按钮时这两个文本框的内容将会被插入数据库。

程序清单：codes\08\8.3\DBTest\src\org\crazyit\db\DBTest.java

```
public class DBTest extends Activity
{
      SQLiteDatabase db;
      Button bn = null;
      ListView listView;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 创建或打开数据库（此处需要使用绝对路径）
            db = SQLiteDatabase.openOrCreateDatabase(
                  this.getFilesDir().toString()
                  + "/my.db3", null); // ①
            listView = (ListView) findViewById(R.id.show);
            bn = (Button) findViewById(R.id.ok);
            bn.setOnClickListener(new OnClickListener()
            {
                  @Override
                  public void onClick(View source)
                  {
                        // 获取用户输入
                        String title = ((EditText) findViewById(
```

```
                        R.id.title)).getText().toString();
                    String content = ((EditText) findViewById(R.id.content))
                        .getText().toString();
                    try
                    {
                        insertData(db, title, content);
                        Cursor cursor = db.rawQuery("select * from news_inf"
                            , null);
                        inflateList(cursor);
                    }
                    catch (SQLiteException se)
                    {
                        // 执行 DDL 创建数据表
                        db.execSQL("create table news_inf(_id integer"
                            + " primary key autoincrement,"
                            + " news_title varchar(50),"
                            + " news_content varchar(255))");
                        // 执行 insert 语句插入数据
                        insertData(db, title, content);
                        // 执行查询
                        Cursor cursor = db.rawQuery("select * from news_inf"
                            , null);
                        inflateList(cursor);
                    }
                }
            });
        }
        private void insertData(SQLiteDatabase db
            , String title, String content) //②
        {
            // 执行插入语句
            db.execSQL("insert into news_inf values(null , ? , ?)"
                , new String[] {title, content });
        }
        private void inflateList(Cursor cursor)
        {
            // 填充 SimpleCursorAdapter
            SimpleCursorAdapter adapter = new SimpleCursorAdapter(
                DBTest.this,
                R.layout.line, cursor,
                new String[] { "news_title", "news_content" }
                , new int[] {R.id.my_title, R.id.my_content },
                CursorAdapter.FLAG_REGISTER_CONTENT_OBSERVER); //③
            // 显示数据
            listView.setAdapter(adapter);
        }
        @Override
        public void onDestroy()
        {
            super.onDestroy();
            // 退出程序时关闭 SQLiteDatabase
            if (db != null && db.isOpen())
            {
                db.close();
            }
        }
    }
```

上面的程序中①号粗体字代码用于创建或打开 SQLite 数据库。当用户单击程序中的“插入”按钮时，程序会调用②号粗体字代码向底层数据表中插入一行记录，并执行查询语句，把底层数据表中的记录查询出来，并使用 ListView 将查询结果（Cursor）显示出来。

程序中③号代码用于将 Cursor 封装成 SimpleCursorAdapter，这个 SimpleCursorAdapter 实现了 Adapter 接口，因此可以作为 ListView 或 GridView 的内容适配器。

SimpleCursorAdapter 的构造器参数与 SimpleAdapter 的构造器参数大致相同，区别是 SimpleAdapter 负责封装集合元素为 Map 的 List，而 SimpleCursorAdapter 负责封装 Cursor——如果我们把 Cursor 里的结果集当成 List 集合，Cursor 里的每一行当成 Map 处理（以数据列的列名为 key，数据列的值为 value），那么 SimpleCursorAdapter 与 SimpleAdapter 就统一起来了。

运行上面程序，看到如图 8.7 所示的界面。



图 8.7　数据库访问示例

正如上面的程序中可看到的，当程序不断地向底层数据表中插入数据时，程序中 ListView 将可以把底层数据表中数据显示出来。

需要指出的是，使用 SimpleCursorAdapter 封装 Cursor 时要求底层数据表的主键列的列名为_id，因为 SimpleCursorAdapter 只能识别列名为_id 的主键。因此上面的程序创建数据表时指定了主键列的列名为_id，否则就会出现 java.lang.IllegalArgumentException: column '_id' does not exist 错误。

上面的程序中我们重写了 Activity 的 onDestroy()方法，当应用程序退出该 Activity 时将会回调该方法，程序在该方法中关闭了 SQLiteDatabase——就像 JDBC 编程中需要关闭 Statement 和 Connection 一样，这里也需要关闭 SQLiteDatabase，否则可能引发资源泄漏。总结起来使用 SQLiteDatabase 进行数据库操作的步骤如下：

1. 获取 SQLiteDatabase 对象，它代表了与数据库的连接。
2. 调用 SQLiteDatabase 的方法来执行 SQL 语句。
3. 操作 SQL 语句的执行结果，比如用 SimpleCursorAdapter 封装 Cursor。
4. 关闭 SQLiteDatabase，回收资源。

8.3.4　使用 sqlite3 工具

在 Android SDK 的 tools 目录下提供了一个 sqlite3.exe 工具，它是一个简单的 SQLite 数据库管理工具，类似于 MySQL 提供的命令行窗口。在有些时候，开发者利用该工具来查询、管理数据库。

例如我们把上面的应用程序所生成的 my.db3 导出到本地计算机上的 F:\盘根目录下，接下来可运行如下命令来启动 SQLite 数据库：

```
sqlite3 f:/my.db3
```

运行上面的命令可以看到如图 8.8 所示的窗口。

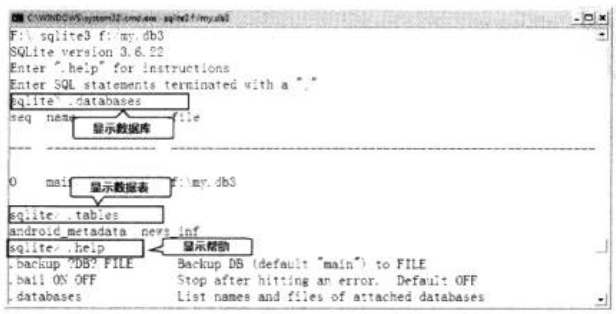


图 8.8　使用 sqlite3 工具

从图 8.8 可以看出，sqlite3 中常用的命令如下。

- .datbases：查看当前数据库。
- .tables：查看当前数据库里的数据表。
- .help：查看 sqlite3 支持的命令。

当然，sqlite3 还支持一些常用的命令，当开发者在图 8.8 所示窗口中输入.help 之后，该工具将会列出这些命令。除此之外，SQLite 数据库还支持绝大部分常用的 SQL 语句，开发者可在图 8.8 所示的窗口运行各种 DDL、DML、查询语句来测试它们。



提示：

SQLite 数据库所支持的 SQL 语句与 MySQL 大致相同，开发者完全可以把已有的 MySQL 经验“移植”到 SQLite 数据库上。当然，当 Android 应用提示某条 SQL 语句有语法错误时，最好先利用 sqlite3 这个工具类测试这条语句、以保证这条 SQL 语句的语法正确。

需要指出的是，SQLite 内部只支持 NULL、INTEGER、REAL（浮点数）、TEXT（文本）和 BLOB（大二进制对象）这 5 种数据类型，但实际上 SQLite 完全可以接受 varchar(n)、char(n)、decimal(p,s) 等数据类型，只不过 SQLite 会在运算或保存时将它们转换为上面 5 种数据类型中相应的类型。

除此之外，SQLite 还有一个特点：它允许把各种类型的数据保存到任何类型字段中，开发者可以不用关心声明该字段所使用的数据类型。例如程序可以把字符串类型的值存入 INTEGER 类型的字段中，也可以把数值类型的值存入布尔类型的字段中……但有一种情况例外：定义为 INTEGER PRIMARY KEY 的字段只能存储 64 位整数， 当向这种字段保存除整数以外的其他类型的数据时，SQLite 会产生错误。

由于 SQLite 允许存入数据时忽略底层数据列实际的数据类型，因此在编写建表语句时可以省略数据列后面的类型声明，例如如下 SQL 语句对于 SQLite 也是正确的：

```
create table my_test
(
    _id integer primary key autoincrement,
    name ,
    pass ,
    gender
);
```

开发者可以把 MySQL 开发经验直接移植到 SQLite 数据库上，如果开发者需要了解更多关于 SQL 语法、MySQL 数据库的知识，请参考疯狂 Java 体系的《疯狂 Java 讲义》一书。

8.3.5 使用特定方法操作 SQLite 数据库

如果开发者对于 SQL 语法不熟悉，甚至以前从未使用过任何数据库，Android 的 SQLiteDatabase 提供了 insert、upate、delete 或 query 语句来操作数据库。

不过既然 Android 提供了这些方法，这里也简单介绍一下。



提示：

虽然 Android 提供了这些所谓的“便捷”方法来操作 SQLite 数据库，但在笔者看来这些方法纯属“鸡肋”，对于一个程序员而言，SQL 语法可以说是基本功中的基本功——你见过不会 1+1=2 的数学工作者吗？

1. 使用 insert 方法插入记录

SQLiteDatabase 的 insert 方法的签名为 long insert (String table, String nullColumnHack, ContentValues values)，这个插入方法的参数说明如下。

- table：代表想插入数据的表名。
- nullColumnHack：代表强行插入 null 值的数据列的列名。当 values 参数为 null 或不包含任何 key-value 对时该参数有效。
- values：代表一行记录的数据。

insert 方法插入的一行记录使用 ContentValues 存放，ContentValues 类似于 Map，它提供了 put(String key, Xxx value)（其中 key 为数据列的列名）方法用于存入数据、getAsXxx(String key)方法用于取出数据。

例如如下语句：

```
ContentValues values = new ContentValues();
values.put("name", "孙悟空");
values.put("age", 500);
// 返回新添记录的行号，该行号是一个内部值，与主键 id 无关，发生错误返回-1
long rowid = db.insert("person_inf", null, values);
```

不管第三个参数是否包含数据，执行 insert()方法总会添加一条记录，如果第三个参数为空，会添加一条除主键之外其他字段值都为 null 的记录。

insert()方法的底层实际上依然是通过构造 insert SQL 语句来进行插入的，因此它生成的 SQL 语句总是形如下面的语句：

```
// ContentValue 里 key-value 对的数量决定了下面的 key-value 对。
insert into <表名>(key1 , key 2 ...)
values(value1 , value2 ...)
```

此时如果第三个参数为 null 或其中 key-value 对的数量为 0，由于 insert()方法还会按此方式生成一条 insert 语句，此时的 insert 语句为：

```
// ContentValue 里 key-value 对的数量决定了下面的 key-value 对。
insert into <表名>()
values()
```

上面的 SQL 语句显然有问题。为了满足 SQL 语法的需要， insert 语句必须给定一个列名，如：insert into person(name) values(null)，这个 name 列名就由第二个参数来指定。由此可见，当 ContentValues 为 null 或它包含的 key-value 对的数量为 0 时，第二个参数就会起作用了。

一般来说，第二个参数指定的列名不应该是主键列的列名，也不应该是非空列的列名，否则强行往这些数据列插入 null 会引发异常。

2. 使用 update 方法更新记录

SQLiteDatabase 的 update 方法的签名为 update(String table, ContentValues values, String whereClause, String[] whereArgs)，这个更新方法的参数说明如下。

- table：代表想更新数据的表名。
- values：代表想更新的数据。
- whereClause：满足该 whereClause 子句的记录将会被更新。
- whereArgs：用于为 whereClause 子句传入参数。

该方法返回受此 update 语句影响的记录的条数。

例如我们想更新 person_inf 表中所有主键大于 20 的人的人名，可调用如下方法：

```
ContentValues values = new ContentValues();
// 存放更新后的人名
values.put("name", "新人名");
int result= db.update("person_inf", values , "_id>?" , new Integer[]{20});
```

实际上 update 方法底层对应的 SQL 语句如下：

```
update <table>
set key1=value1 , key2=value2 ...
where <whereClause>
```

其中 whereArgs 参数用于向 whereClause 中传入参数。

3. 使用 delete 方法删除记录

SQLiteDatabase 的 delete 方法的签名为 delete(String table , String whereClause, String[] whereArgs)，这个删除的参数说明如下。

- table：代表想删除数据的表名。
- whereClause：满足该 whereClause 子句的记录将会被删除。
- whereArgs：用于为 whereClause 子句传入参数。

该方法返回受此 delete 语句影响的记录的条数。

例如我们想删除 person_inf 表中所有人名以“孙”开头的记录，可调用如下方法：

```
int result= db.delete("person_inf", "person_name like ?" , new String[]{"孙_"});
```

实际上 delete 方法底层对应的 SQL 语句如下：

```
delete <table>
where <whereClause>
```

4. 使用 query 方法查询记录

SQLiteDatabase 的 query 方法的签名为 Cursor query(boolean distinct, String table, String[] columns, String whereClause, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)，这个 query 方法的参数说明如下。

- distinct：指定是否去除重复记录。
- table：执行查询数据的表名。
- columns：要查询出来的列名。相当于 select 语句 select 关键字后面的部分。
- whereClause：查询条件子句，相当于 select 语句 where 关键字后面的部分，在条件子句中允许使用占位符“?”。
- whereArgs：用于为 whereClause 子句中占位符传入参数值，值在数组中的位置与占位符在语句中的位置必须一致，否则就会有异常。
- groupBy：用于控制分组。相当于 select 语句 group by 关键字后面的部分
- having：用于对分组进行过滤。相当于 select 语句 having 关键字后面的部分
- orderBy：用于对记录进行排序。相当于 select 语句 order by 关键字后面的部分，如：personid desc, age asc;
- limit：用于进行分页，相当于 select 语句 limit 关键字后面的部分。例如 5,10。

看到这个方法的设计，笔者忍不住想再次表达对这个方法的不满：如果读者完全不懂

SQL 语句，那需要花多少时间才能理解这个方法中这么多参数的设置？如果读者愿意花时间去理解这个方法中各参数的设置，那么所花的时间已经足够去掌握这条 select 语句的语法格式了。

当然，这个 query()方法也并非完全一无是处：当应用程序需要进行“条件不确定”的查询（即查询条件需要动态改变的查询）时，使用这个 query 方法可以避免手动拼接 SQL 语句。

例如想查询出 person_inf 表中人名以“孙”开头的记录，可使用如下语句：

```
Cursor cursor = db.query("person_inf", new String[]{"_id,name,age"}
      , "name like ?", new String[]{"孙%"}
      , null, null, "personid desc", "5,10");
// 处理结果集
cursor.close();
```

8.3.6 事务

SQLiteDatabase 中包含如下两个方法来控制事务。

- beginTransaction()：开始事务。
- endTransaction()：结束事务。

除此之外，SQLiteDatabase 还提供了如下方法来判断当前上下文是否处于事务环境中。

- inTransaction()：如果当前上下文处于事务中，则返回 true；否则返回 false。

当程序执行 endTransaction()方法时将会结束事务——那到底是提交事务呢，还是回滚事务呢？这取决于 SQLiteDatabase 是否调用了 setTransactionSuccessful()方法来设置事务标志，如果程序事务执行中调用该方法设置了事务成功则提交事务；否则程序将会回滚事务。

示例代码如下：

```
// 开始事务
db.beginTransaction();
try
{
      // 执行 DML 语句
      ...
      // 调用该方法设置事务成功；否则 endTransaction()方法将回滚事务
      db.setTransactionSuccessful();
}
finally
{
      // 由事务的标志决定是提交事务还是回滚事务
      db.endTransaction();
}
```

8.3.7 SQLiteOpenHelper 类

在上一个示例程序中，我们为了判断底层数据库是否包含 news_inf 数据表，采用的处理方法十分烦琐：程序先尝试向 news_inf 数据表中插入记录，如果程序抛出异常，在异常捕获的 catch 块中创建 news_inf 数据表、然后再插入记录。那么到底是否有一种更优雅的方式来处理这种问题呢？有，Android 提供了 SQLiteOpenHelper 类来处理这个问题。



提示：

实际项目中很少使用 SQLiteDatabase 的方法来打开数据库，通常都会继承 SQLiteOpenHelper 开发子类，并通过该子类的 getReadableDatabase()、getWritableDatabase()方法打开数据库。

SQLiteOpenHelper 是 Android 提供的一个管理数据库的工具类，可用于管理数据库的创建和版本更新。一般的用法是创建 SQLiteOpenHelper 的子类，并扩展它的 onCreate（SQLiteDatabase db）和 onUpgrade（SQLiteDatabase db, int oldVersion, int newVersion）方法。

SQLiteOpenHelper 包含如下常用的方法。

- synchronized SQLiteDatabase getReadableDatabase()：以读写的方式打开数据库对应的 SQLiteDatabase 对象。
- synchronized SQLiteDatabase getWritableDatabase()：以写的方式打开数据库对应的 SQLiteDatabase 对象。
- abstract void onCreate（SQLiteDatabase db）：当第一次创建数据库时回调该方法。
- abstract void onUpgrade（SQLiteDatabase db, int oldVersion, int newVersion）：当数据库版本更新时回调该方法。
- synchronized void close()：关闭所有打开的 SQLiteDatabase。

从上面的方法介绍中不难看出，SQLiteOpenHelper 提供了 getReadableDatabase()、getWritable Database()两个方法用于打开数据库连接，并提供了 close 方法来关闭数据库连接，而开发者需要做的就是重写它的两个抽象方法。

- onCreate（SQLiteDatabase db）：用于初次使用软件时生成数据库表，当调用 SQLiteOpenHelper 的 getWritableDatabase()或者 getReadableDatabase()方法获取用于操作数据库的 SQLiteDatabase 实例时，如果数据库不存在，Android 系统会自动生成一个数据库，接着调用 onCreate()方法，onCreate()方法在初次生成数据库时才会被调用，重写 onCreate()方法时，可以生成数据库表结构及添加一些应用使用到的初始化数据。
- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)：用于升级软件时更新数据库表结构，方法在数据库的版本发生变化时会被调用，该方法调用时 oldVersion 代表数据库之前的版本号，newVersion 代表当前数据库当前的版本号。那么在哪里指定数据库的版本号呢？当程序创建 SQLiteOpenHelper 对象时，必须指定一个 version 参数，该参数就决定了所使用的数据库的版本——也就是说，数据库的版本是由程序员控制的。只要某次创建 SQLiteOpenHelper 时指定的数据库版本号高于之前指定的版本号，系统就会自动触发 onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)方法，程序就可以在 onUpgrade()方法里面根据原版号和目标版本号进行判断，即可根据版本号进行必需的表结构更新。



提示：

实际上，当应用程序升级表结构时，完全可能因为已有的数据导致升级失败。在这种时候程序可能需要先对数据进行转储，清空数据表中的记录，接着对数据表进行更新，当数据表更新完成后再将数据保存回来。

一旦得到了 SQLiteOpenHelper 对象之后，程序无须使用 SQLiteDatabase 的静态方法创建 SQLiteDatabase 实例，而且可以使用 getWritableDatabase()或 getReadableDatabase()方法来获取一个用于操作数据库的 SQLiteDatabase 实例。

其中 getWritableDatabase() 方法以读写方式打开数据库，一旦数据库的磁盘空间满了，数据库就只能读而不能写，倘若使用 getWritableDatabase()打开数据库就会出错。getReadableDatabase()方法先以读写方式打开数据库，如果数据库的磁盘空间满了，就会打开

失败，当打开失败后会继续尝试以只读方式打开数据库。

下面以一个实例程序来说明 SQLiteOpenHelper 的功能与用法。

实例：英文生词本

该实例允许用户将自己不熟悉的单词添加到系统数据库中，当用户需要查询某个单词或解释时，只要在程序中输入相应的关键词，程序中相应的条目就会显示出来。

该程序使用了 SQLiteOpenHelper 的子类来管理数据库连接，当程序通过 SQLiteOpenHelper 的子类来获取数据库连接时，如果数据库还不存在，系统会自动调用 onCreate(SQLiteDatabase db)方法来创建数据库；如果程序创建 SQLiteOpenHelper 的子类的实例时传入的数据库版本号高于之前的版本号，系统会自动调用 onUpgrade(SQLite Database db, int oldVersion, int newVersion)方法来更新数据库。

下面是 SQLiteOpenHelper 的子类的代码。

程序清单：codes\08\8.3\Dict\src\org\crazyit\db\MyDatabaseHelper.java

```
public class MyDatabaseHelper extends SQLiteOpenHelper
{
    final String CREATE_TABLE_SQL =
        "create table dict(_id integer primary "" +
        "key autoincrement , word , detail)";
    public MyDatabaseHelper(Context context, String name, int version)
    {
        super(context, name, null, version);
    }
    @Override
    public void onCreate(SQLiteDatabase db)
    {
        // 第一次使用数据库时自动建表
        db.execSQL(CREATE_TABLE_SQL);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db
        , int oldVersion, int newVersion)
    {
        System.out.println("--------onUpdate Called--------"
            + oldVersion + "--->" + newVersion);
    }
}
```

上面的 MyDatabaseHelper 继承了 SQLiteOpenHelper，并重写了其基类的 onCreate (SQLiteDatabase db)方法，该方法中执行的建表语句用于初始化系统数据表。如果用户第一次使用该程序，系统将会自动调用 onCreate(SQLiteDatabase db)方法来初始化底层数据库。



图 8.9　生词本界面

MyDatabaseHelper 工具类的作用主要是管理数据库的初始化，并允许应用程序通过该工具类来获取 SQLiteDatabase 对象。接下来的程序就可通过该工具类获取 SQLiteDatabase 对象，并利用该对象操作数据库。

本实例的程序界面比较简单，它提供了输入框让用户添加生词，并提供输入框让用户输入要查询的关键字，用户在查询文本框中输入查询关键字后单击“查询”按钮即可显示相应的生词条目。该程序的界面如图 8.9 所示。

图 8.9 所示的界面中前两个文本框用户添加生词；第三个文本框用

户输入想查询的关键词，当用户输入相应的关键词，并单击“查找”按钮后，系统将会查询相应的条目。下面是该生词本的主程序。

程序清单：codes\08\8.3\Dict\src\org\crazyit\db\Dict.java

```
public class Dict extends Activity
{
    MyDatabaseHelper dbHelper;
    Button insert = null;
    Button search = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建 MyDatabaseHelper 对象，指定数据库版本为 1，此处使用相对路径即可
        // 数据库文件自动会保存在程序的数据文件夹的 databases 目录下
        dbHelper = new MyDatabaseHelper(this, "myDict.db3", 1);
        insert = (Button) findViewById(R.id.insert);
        search = (Button) findViewById(R.id.search);
        insert.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String word = ((EditText) findViewById(R.id.word))
                        .getText().toString();
                String detail = ((EditText) findViewById(R.id.detail))
                        .getText().toString();
                // 插入生词记录
                insertData(dbHelper.getReadableDatabase(), word, detail);
                // 显示提示信息
                Toast.makeText(Dict.this, "添加生词成功！", 8000).show();
            }
        });
        search.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String key = ((EditText) findViewById(R.id.key)).getText()
                        .toString();
                // 执行查询
                Cursor cursor = dbHelper.getReadableDatabase().rawQuery(
                        "select * from dict where word like ? or detail like ?",
                        new String[] { "%" + key + "%", "%" + key + "%" });
                // 创建一个 Bundle 对象
                Bundle data = new Bundle();
                data.putSerializable("data", converCursorToList(cursor));
                // 创建一个 Intent
                Intent intent = new Intent(Dict.this
                        , ResultActivity.class);
                intent.putExtras(data);
                // 启动 Activity
                startActivity(intent);
            }
        });
    }
    protected ArrayList<Map<String, String>>
```

```
        converCursorToList(Cursor cursor)
    {
        ArrayList<Map<String, String>> result =
            new ArrayList<Map<String, String>>();
        // 遍历 Cursor 结果集
        while (cursor.moveToNext())
        {
            // 将结果集中的数据存入 ArrayList 中
            Map<String, String> map = new HashMap<String, String>();
            // 取出查询记录中第 2 列、第 3 列的值
            map.put("word", cursor.getString(1));
            map.put("detail", cursor.getString(2));
            result.add(map);
        }
        return result;
    }
    private void insertData(SQLiteDatabase db, String word
        , String detail)
    {
        // 执行插入语句
        db.execSQL("insert into dict values(null , ? , ?)"
            , new String[] {word, detail });
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 退出程序时关闭 MyDatabaseHelper 里的 SQLiteDatabase
        if (dbHelper != null)
        {
            dbHelper.close();
        }
    }
}
```

上面的程序中前两行粗体字代码用于根据 SQLiteOpenHelper 所获取的 SQLiteDatabase 来执行插入数据、查询数据。程序重写了 Activity 的 onDestroy()方法，并在该方法中调用 SQLiteOpenHelper 的 close()方法来关闭数据库连接。

当程序第一次调用 getWritableDatabase() 或 getReadableDatabase() 方法后，SQLiteOpenHelper 会缓存已获得的 SQLiteDatabase 实例，SQLiteDatabase 实例正常情况下会维持数据库的打开状态，因此程序退出时应该关闭不再使用的 SQLiteDatabase。一旦 SQLiteOpenHelper 缓存了 SQLiteDatabase 实例之后，多次调用 getWritableDatabase()或 getReadableDatabase()方法得到的都是同一个 SQLiteDatabase 实例。

上面的程序执行查询后并未在该 Activity 中显示查询得到的条目，而是使用另一个 Activity：ResultActivity 来显示查询结果，ResultActivity 只是提供了一个 ListView 来显示查询得到的条目。为了把 Dict 查询得到的条目传给 ResultActivity，程序将查询得到的条目封装成 List<Map<String , String>>后传给 ResultActivity，ResultActivity 就可利用 ListView 来显示查询结果了。

下面是 ResultActivity 的代码。

程序清单：codes\08\8.3\Dict\src\org\crazyit\db\ResultActivity.java

```
public class ResultActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
```

```
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.popup);
        ListView listView = (ListView) findViewById(R.id.show);
        Intent intent = getIntent();
        // 获取该 intent 所携带的数据
        Bundle data = intent.getExtras();
        // 从 Bundle 数据包中取出数据
        @SuppressWarnings("unchecked")
        List<Map<String, String>> list = (List<Map<String, String>>)
            data.getSerializable("data");
        // 将 List 封装成 SimpleAdapter
        SimpleAdapter adapter = new SimpleAdapter(ResultActivity.this
            , list,
            R.layout.line, new String[] { "word", "detail" }
            , new int[] {R.id.word, R.id.detail });
        // 填充 ListView
        listView.setAdapter(adapter);
    }
}
```

上面的 ResultActivity 只是一个普通的 Activity，但我们在 AndroidManifest.xml 文件中将该 Activity 设为对话框风格的 Activity，这样就可让应用程序以对话框来显示查询结果。查询结果如图 8.10 所示。



图 8.10 查询生词

8.4 手势（Gesture）

所谓手势，其实是指用户手指或触摸笔在触摸屏上的连续触碰行为，比如在屏幕上从左至右划出的一个动作，就是手势，再比如在屏幕上画出一个圆圈也是手势。手势这种连续的触碰会形成某个方向上的移动趋势，也会形成一个不规则的几何图形。Android 对两种手势行为都提供了支持：

- 对于第一种手势行为而言，Android 提供了手势检测，并为手势检测提供了相应的监听器。
- 对于第二种手势行为，Android 允许开发者添加手势，并提供了相应的 API 识别用户手势。

8.4.1 手势检测

Android 为手势检测提供了一个 GestureDetector 类，GestureDetector 实例代表了一个手势检测器，创建 GestureDetector 时需要传入一个 GestureDetector.OnGestureListener 实例，GestureDetector. OnGestureListener 就是一个监听器、负责对用户的手势行为提供响应。

GestureDetector.OnGestureListener 里包含的事件处理方法如下。

- boolean onDown(MotionEvent e)：当触碰事件按下时触发该方法。
- boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)：当用户在触摸屏上“拖过”时触发该方法。其中 velocityX、velocityY 代表“拖过”动作在横向、纵向上的速度。
- abstract void onLongPress(MotionEvent e)：当用户在屏幕上长按时触发该方法。
- boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)：当用户在屏幕上“滚动”时触发该方法。

- void onShowPress(MotionEvent e)：当用户在触摸屏上按下、而且还未移动和松开时触发该方法。
- boolean onSingleTapUp(MotionEvent e)：用户在触摸屏上的轻击事件将会触发该方法。

关于 GestureDetector.OnGestureListener 监听器里各方法的触发时机，仅从文字上表述总显得比较抽象而且难于理解，下面将以一个最简单的例子来让读者理解各方法的触发时机。

使用 Android 的手势检测只需两个步骤。

① 创建一个 GestureDetector 对象。创建该对象时必须实现一个 GestureDetector.OnGestureListener 监听器实例。

② 为应用程序的 Activity（偶尔也可为特定组件）的 TouchEvent 事件绑定监听器，在事件处理中指定把 Activity（或特定组件）上的 TouchEvent 事件交给 GestureDetector 处理。

经过上面两个步骤之后，Activity（或特定组件）上的 TouchEvent 事件就会交给 GestureDetector 处理，而 GestureDetector 就会检测是否触发了特定的手势动作。

下面的程序测试了用户的不同动作到底触发哪种手势动作。

程序清单：codes\08\8.4\GestureTest\src\org\crazyit\io\GestureTest.java

```
public class GestureTest extends Activity
    implements OnGestureListener
{
    // 定义手势检测器实例
    GestureDetector detector;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //创建手势检测器
        detector = new GestureDetector(this, this);
    }
    //将该 Activity 上的触碰事件交给 GestureDetector 处理
    @Override
    public boolean onTouchEvent(MotionEvent me)
    {
        return detector.onTouchEvent(me);
    }
    @Override
    public boolean onDown(MotionEvent arg0)
    {
        Toast.makeText(this,"onDown"
            , Toast.LENGTH_LONG).show();
        return false;
    }
    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2
        , float velocityX, float velocityY)
    {
        Toast.makeText(this , "onFling"
            , Toast.LENGTH_LONG).show();
        return false;
    }
    @Override
    public void onLongPress(MotionEvent e)
    {
        Toast.makeText(this ,"onLongPress"
            , Toast.LENGTH_LONG).show();
```

```
        }
        @Override
        public boolean onScroll(MotionEvent e1, MotionEvent e2
                , float distanceX, float distanceY)
        {
                Toast.makeText(this ,"onScroll" ,
                        Toast.LENGTH_LONG).show();
                return false;
        }
        @Override
        public void onShowPress(MotionEvent e)
        {
                Toast.makeText(this ,"onShowPress"
                        , Toast.LENGTH_LONG).show();
        }
        @Override
        public boolean onSingleTapUp(MotionEvent e)
        {
                Toast.makeText(this ,"onSingleTapUp"
                        , Toast.LENGTH_LONG).show();
                return false;
        }
}
```

上面的程序中第一行粗体字代码创建了一个 GestureDetector 对象，创建该对象时传入了 this 作为参数，这表明该 Activity 本身将会作为 GestureDetector.On GestureListener 监听器，所以该 Activity 实现了该接口，并实现了该接口里的全部方法。程序中第二行粗体字指定把 Activity 上的 TouchEvent 交给 GestureDetector 处理。

运行上面的程序，当用户随意地在屏幕上触碰时，程序将会检测到用户到底执行了哪些手势，如图 8.11 所示。



图 8.11 手势检测

掌握了用户在屏幕上的哪些动作对应于哪些手势之后，接下来再以两个示例来介绍 Android 手势检测在实际项目中的应用。

实例：通过手势缩放图片

前面介绍过图片缩放的技术实现，通过 Matrix 即可实现图片缩放，但之前的图片缩放要么通过按钮控制，要么通过 SeekBar 来控制图片缩放，这些缩放方式太“传统”了。而本示例所介绍的图片缩放则“炫”得多：用户只要在图片上随意地“挥动”手指，图片就可被缩放——从左向右挥动时图片被放大，当从右向左地挥动时图片被缩小；挥动速度越快，缩放比越大。

该示例的程序界面布局很简单，只在界面中间定义一个 ImageView 来显示图片即可。该程序的思路是使用一个 GestureDetector 来检测用户的手势，并根据用户手势在横向的速度缩放图片。该程序代码如下。

程序清单：codes\08\8.4\GestureZoom\src\org\crazyit\io\GestureZoom.java

```
public class GestureZoom extends Activity
        implements OnGestureListener
{
        // 定义手势检测器实例
        GestureDetector detector;
        ImageView imageView;
        // 初始的图片资源
        Bitmap bitmap;
```

```
// 定义图片的宽、高
int width, height;
// 记录当前的缩放比
float currentScale = 1;
// 控制图片缩放的 Matrix 对象
Matrix matrix;
@Override
public void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      // 创建手势检测器
      detector = new GestureDetector(this, this);
      imageView = (ImageView) findViewById(R.id.show);
      matrix = new Matrix();
      // 获取被缩放的源图片
      bitmap = BitmapFactory.decodeResource(
            this.getResources(), R.drawable.flower);
      // 获得位图宽
      width = bitmap.getWidth();
      // 获得位图高
      height = bitmap.getHeight();
      // 设置 ImageView 初始化时显示的图片。
      imageView.setImageBitmap(BitmapFactory.decodeResource(
            this.getResources(), R.drawable.flower));
}
@Override
public boolean onTouchEvent(MotionEvent me)
{
      // 将该 Activity 上的触碰事件交给 GestureDetector 处理
      return detector.onTouchEvent(me);
}
@Override
public boolean onFling(MotionEvent event1, MotionEvent event2,
      float velocityX, float velocityY) //②
{
      velocityX = velocityX > 4000 ? 4000 : velocityX;
      velocityX = velocityX < -4000 ? -4000 : velocityX;
      // 根据手势的速度来计算缩放比，如果 velocityX>0，放大图像，否则缩小图像
      currentScale += currentScale * velocityX / 4000.0f;
      // 保证 currentScale 不会等于 0
      currentScale = currentScale > 0.01 ? currentScale: 0.01f;
      // 重置 Matrix
      matrix.reset();
      // 缩放 Matrix
      matrix.setScale(currentScale, currentScale, 160, 200);
      BitmapDrawable tmp = (BitmapDrawable)
            imageView.getDrawable();
      // 如果图片还未回收，先强制回收该图片
      if (!tmp.getBitmap().isRecycled()) // ①
      {
            tmp.getBitmap().recycle();
      }
      // 根据原始位图和 Matrix 创建新图片
      Bitmap bitmap2 = Bitmap.createBitmap(bitmap, 0, 0
            , width, height, matrix, true);
      // 显示新的位图
      imageView.setImageBitmap(bitmap2);
      return true;
}
@Override
```

```
        public boolean onDown(MotionEvent arg0)
        {
            return false;
        }
        @Override
        public void onLongPress(MotionEvent event)
        {
        }
        @Override
        public boolean onScroll(MotionEvent event1
            , MotionEvent event2, float distanceX, float distanceY)
        {
            return false;
        }
        @Override
        public void onShowPress(MotionEvent event)
        {
        }
        @Override
        public boolean onSingleTapUp(MotionEvent event)
        {
            return false;
        }
}
```

上面的程序的处理方式与前一个程序的处理方法如出一辙：第一行粗体字代码用于创建 GestureDetector 对象，第二行粗体字代码指定把 Activity 的 OnTouch 事件交给 GestureDetector 处理。

该程序与前一个程序的不同之处在于：程序实现 GestureDetector.OnGestureListener 监听器时，只实现了②号代码所标出的 onFling(MotionEvent event1, MotionEvent event2,float velocityX, float velocityY)方法，并在该方法内根据 velocityX 参数（横向上的拖动速度）来计算图片上缩放比，这样该程序即可根据用户的“手势”来缩放图片。

实例：通过手势实现翻页效果

本示例的手势检测思路还是一样：把 Activity 的 TouchEvent 交给 GestureDetector 处理。这个程序的特殊之处在于，该程序使用了一个 ViewFlipper 组件，ViewFlipper 可使用动画控制多个组件之间的切换效果。

本程序通过 GestureDetector 来检测用户的手势动作，并根据手势动作来控制 ViewFlipper 包含的 View 组件的切换，从而实现翻页效果。

该程序的界面布局代码如下。

程序清单：codes\08\8.4\GestureFlip\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<!-- 定义 ViewFlipper 组件 -->
<ViewFlipper android:id="@+id/flipper"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    />
</LinearLayout>
```

上面的界面布局文件中定义了一个 ViewFlipper 组件，该组件可控制多个 View 的动画切换。

该示例的程序代码如下。

程序清单：codes\08\8.4\GestureFlip\src\org\crazyit\io\GestureFlip.java

```
public class GestureFlip extends Activity
    implements OnGestureListener
{
    // ViewFlipper 实例
    ViewFlipper flipper;
    // 定义手势检测器实例
    GestureDetector detector;
    // 定义一个动画数组，用于为 ViewFlipper 指定切换动画效果
    Animation[] animations = new Animation[4];
    // 定义手势动作两点之间的最小距离
    final int FLIP_DISTANCE = 50;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建手势检测器
        detector = new GestureDetector(this, this);
        // 获得 ViewFlipper 实例
        flipper = (ViewFlipper) this.findViewById(R.id.flipper);
        // 为 ViewFlipper 添加 5 个 ImageView 组件
        flipper.addView(addImageView(R.drawable.java));
        flipper.addView(addImageView(R.drawable.ee));
        flipper.addView(addImageView(R.drawable.ajax));
        flipper.addView(addImageView(R.drawable.xml));
        flipper.addView(addImageView(R.drawable.classic));
        // 初始化 Animation 数组
        animations[0] = AnimationUtils.loadAnimation(
            this, R.anim.left_in);
        animations[1] = AnimationUtils.loadAnimation(
            this, R.anim.left_out);
        animations[2] = AnimationUtils.loadAnimation(
            this, R.anim.right_in);
        animations[3] = AnimationUtils.loadAnimation(
            this, R.anim.right_out);
    }
    // 定义添加 ImageView 的工具方法
    private View addImageView(int resId)
    {
        ImageView imageView = new ImageView(this);
        imageView.setImageResource(resId);
        imageView.setScaleType(ImageView.ScaleType.CENTER);
        return imageView;
    }
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent event2,
        float velocityX, float velocityY)
    {
        // 如果第一个触点事件的 X 坐标大于第二个触点事件的 X 坐标超过 FLIP_DISTANCE
        // 也就是手势从右向左滑
        if (event1.getX() - event2.getX() > FLIP_DISTANCE)
        {
            // 为 flipper 设置切换的的动画效果
            flipper.setInAnimation(animations[0]);
            flipper.setOutAnimation(animations[1]);
```

```
                flipper.showPrevious();
                return true;
            }
            // 如果第二个触点事件的 X 坐标大于第一个触点事件的 X 坐标超过 FLIP_DISTANCE
            // 也就是手势从右向左滑
            else if (event2.getX() - event1.getX() > FLIP_DISTANCE)
            {
                // 为 flipper 设置切换的动画效果
                flipper.setInAnimation(animations[2]);
                flipper.setOutAnimation(animations[3]);
                flipper.showNext();
                return true;
            }
            return false;
        }
        @Override
        public boolean onTouchEvent(MotionEvent event)
        {
            // 将该 Activity 上的触碰事件交给 GestureDetector 处理
            return detector.onTouchEvent(event);
        }
        @Override
        public boolean onDown(MotionEvent arg0)
        {
            return false;
        }
        @Override
        public void onLongPress(MotionEvent event)
        {
        }
        @Override
        public boolean onScroll(MotionEvent event1
                , MotionEvent event2, float arg2, float arg3)
        {
            return false;
        }
        @Override
        public void onShowPress(MotionEvent event)
        {
        }
        @Override
        public boolean onSingleTapUp(MotionEvent event)
        {
            return false;
        }
    }
```

该程序同样只是实现了 GestureDetector.OnGestureListener 的 onFling 方法，上面的程序中粗体字代码负责实现：当 event1.getX() - event2.getX()的距离大于特定距离时，即可判断用户手势为从右向左滑动，此时设置 ViewFlipper 采用动画方式切换为上一个 View；当 event2.getX() - event1.getX()的距离大于特定距离时，即可判断用户手势为从左向右滑动，此时设置 ViewFlipper 采用动画方式切换为下一个 View——这样就实现了所谓“翻书”效果。

运行上面的程序，并在屏幕上执行“拖动”手势，即可看到 ViewFlipper 内组件切换的效果，如图 8.12 所示。



图 8.12 采用手势检测实现翻页效果

8.4.2 增加手势

Android 除了提供了手势检测之外，还允许应用程序把用户手势（多个持续的触摸事件在屏幕上形成特定的形状）添加到指定文件中，以备以后使用——如果程序需要，当用户下次再次画出该手势时，系统将可识别该手势。

Android 使用 GestureLibrary 来代表手势库，并提供了 GestureLibraries 工具类来创建手势库，GestureLibraries 提供了如下 4 个静态方法从不同位置加载手势库。

- static GestureLibrary fromFile(String path)：从 path 代表的文件中加载手势库。
- static GestureLibrary fromFile(File path)：从 path 代表的文件中加载手势库。
- static GestureLibrary fromPrivateFile(Context context, String name)：从指定应用程序的数据文件夹中 name 文件中加载手势库。
- static GestureLibrary fromRawResource(Context context, int resourceId)：从 resourceId 所代表的资源中加载手势库。

一旦在程序中获得了 GestureLibrary 对象之后，该对象提供了如下方法来添加手势、识别手势。

- void addGesture(String entryName, Gesture gesture)：添加一个名为 entryName 的手势。
- Set<String> getGestureEntries()：获取该手势库中的所有手势的名称。
- ArrayList<Gesture> getGestures(String entryName)：获取 entryName 名称对应的全部手势。
- ArrayList<Prediction> recognize(Gesture gesture)：从当前手势库中识别与 gesture 匹配的全部手势。
- void removeEntry(String entryName)：删除手势库中 entryName 对应的手势。
- void removeGesture(String entryName, Gesture gesture)：删除手势库中 entryName、gesture 对应的手势。
- boolean save()：当向手势库中添加手势或从中删除手势后调用该方法保存手势库。

Android 除了提供了 GestureLibraries、GestureLibrary 来管理手势之外，还提供了一个专门的手势编辑组件：GestureOverlayView，该组件就像一个“绘图组件”，只是用户在组件上绘制的不是图形，而是手势。

为了监听 GestureOverlayView 组件上的手势事件，Android 为 GestureOverlayView 提供了 OnGestureListener、OnGesturePerformedListener、OnGesturingListener 三个监听器接口，这些监听器所包含的方法分别用于响应手势事件开始、结束、完成、取消等事件，开发者可根据实际需要来选择不同的监听器——一般来说，OnGesturePerformedListener 是最常用的监听器，它可用于在手势事件完成时提供响应。

下面的应用程序在界面布局中使用了 GestureOverlayView。

程序清单：codes\08\8.4\AddGesture\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal"
    >
```

```
<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="请在下面屏幕上绘制手势"/>
<!-- 使用手势绘制组件 -->
<android.gesture.GestureOverlayView
      android:id="@+id/gesture"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gestureStrokeType="multiple" />
</LinearLayout>
```

由于 GestureOverlayView 并不是标准的视图组件，因此在界面布局中使用该组件时需要使用全限定的类名。

上面的程序中使用 GestureOverlayView 组件时指定了一个 android:gestureStrokeType 参数，该参数控制手势是否需要多一笔完成。大部分时候，一个手势只要一笔就可以完成，此时可将该参数设为 single。如果该手势需要多笔来完成，则将该参数设为 multiple。

接下来程序将会为 GestureOverlayView 添加一个 OnGesturePerformedListener 监听器，当手势事件完成时，该监听器会打开一个对话框，让用户选择保存该手势。程序代码如下。

程序清单：codes\08\8.4\AddGesture\src\org\crazyit\io\AddGesture.java

```
public class AddGesture extends Activity
{
      EditText editText;
      GestureOverlayView gestureView;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取文本编辑框
            editText = (EditText) findViewById(R.id.gesture_name);
            // 获取手势编辑视图
            gestureView = (GestureOverlayView)
                  findViewById(R.id.gesture);
            // 设置手势的绘制颜色
            gestureView.setGestureColor(Color.RED);
            // 设置手势的绘制宽度
            gestureView.setGestureStrokeWidth(4);
            // 为 gesture 的手势完成事件绑定事件监听器
            gestureView.addOnGesturePerformedListener(
                  new OnGesturePerformedListener()
                  {
                        @Override
                        public void onGesturePerformed(GestureOverlayView overlay,
                              final Gesture gesture)
                        {
                              // 加载 save.xml 界面布局代表的视图
                              View saveDialog = getLayoutInflater().inflate(
                                    R.layout.save, null);
                              // 获取 saveDialog 里的 show 组件
                              ImageView imageView = (ImageView) saveDialog
                                    .findViewById(R.id.show);
                              // 获取 saveDialog 里的 gesture_name 组件
                              final EditText gestureName = (EditText) saveDialog
                                    .findViewById(R.id.gesture_name);
                              // 根据 Gesture 包含的手势创建一个位图
                              Bitmap bitmap = gesture.toBitmap(128,
                                    128, 10, 0xffff0000);
```

```
                                imageView.setImageBitmap(bitmap);
                                // 使用对话框显示 saveDialog 组件
                                new AlertDialog.Builder(AddGesture.this)
                                .setView(saveDialog)
                                .setPositiveButton("保存", new OnClickListener()
                                {
                                    @Override
                                    public void onClick(DialogInterface dialog,
                                        int which)
                                    {
                                        // 获取指定文件对应的手势库
                                        GestureLibrary gestureLib = GestureLibraries
                                            .fromFile("/mnt/sdcard/mygestures");
                                        // 添加手势
                                        gestureLib.addGesture(gestureName.getText()
                                            .toString(), gesture);
                                        // 保存手势库
                                        gestureLib.save();
                                    }
                                }).setNegativeButton("取消", null).show();
                            }
                        });
                }
            }
```

上面的程序中第一行粗体字代码用于为 GestureOverlayView 绑定 OnGesturePerformed Listener 监听器，该监听器用于在手势完成时提供响应——它的响应就是打开一个对话框。该对话框的界面布局代码如下。

程序清单：codes\08\8.4\AddGesture\res\layout\save.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginRight="8dip"
    android:text="@string/gesture_name"
     />
<!-- 定义一个文本框来让用户输入手势名 -->
<EditText
    android:id="@+id/gesture_name"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
</LinearLayout>
<!-- 定义一个图片框来显示手势 -->
<ImageView
    android:id="@+id/show"
    android:layout_width="128dp"
    android:layout_height="128dp"
    android:layout_marginTop="10dp" />
</LinearLayout>
```

AddGesture 程序中的第二段粗体字代码是于对话框中完成的，这段粗体字代码用于从 SD

卡的指定文件中加载手势库，并添加用户刚刚输入的手势。

运行该程序将看到如图 8.13 所示的界面。

用户可在图 8.13 所示的程序中随意地“绘制”手势，绘制完成后 OnGesturePerformedListener 监听器将会打开如图 8.14 所示的对话框。



图 8.13 编辑手势



图 8.14 保存手势

当用户单击图 8.14 所示对话框中的“保存”按钮后，程序将会调用 GestureLibrary 的 addGesture 方法来添加手势，并调用 save()方法来保存手势。

一旦用户通过该程序建立了自己的手势库，接下来就可在其他程序中使用该手势库了。

注意：

上面的程序需要将手势库保存在 SD 卡上，因此还需要授予该应用程序读、写 SD 卡的权限。



8.4.3 识别用户的手势

前面已经提到，GestureLibrary 提供了 recognize(Gesture ges)方法来识别手势，该方法将会返回该手势库中所有与 ges 匹配的手势——两个手势的图形越相似，相似度越高。

recognize(Gesture ges)方法返回值为 ArrayList<Prediction>，其中 Prediction 封装了手势的匹配信息，Prediction 对象的 name 属性代表了匹配的手势名，score 属性代表了手势的相似度。

下面的程序将会利用前一个程序所创建的手势库来识别手势，该程序的界面很简单，只是在界面中定义一个 GestureOverlayView 组件、允许用户在该组件上输入手势，程序为该组件绑定了 OnGesturePerformedListener 监听器，该监听器检测到用户手势完成时，就会调用手势库来识别用户输入的手势。

该程序的代码如下。

程序清单：codes\08\8.4\RecogniseGesture\src\org\crazyit\io\RecogniseGesture.java

```
public class RecogniseGesture extends Activity
{
    // 定义手势编辑组件
    GestureOverlayView gestureView;
    // 记录手机上已有的手势库
    GestureLibrary gestureLibrary;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 读取上一个程序所创建的手势库
        gestureLibrary = GestureLibraries
```

```
            .fromFile("/mnt/sdcard/mygestures");
        if (gestureLibrary.load())
        {
            Toast.makeText(RecogniseGesture.this, "手势文件装载成功！",
                Toast.LENGTH_LONG).show();
        }
        else
        {
            Toast.makeText(RecogniseGesture.this, "手势文件装载失败！",
                Toast.LENGTH_LONG).show();
        }
        // 获取手势编辑组件
        gestureView = (GestureOverlayView) findViewById(R.id.gesture);
        // 为手势编辑组件绑定事件监听器
        gestureView.addOnGesturePerformedListener(
            new OnGesturePerformedListener()
            {
                @Override
                public void onGesturePerformed(GestureOverlayView
                    overlay, Gesture gesture)
                {
                    // 识别用户刚刚所绘制的手势
                    ArrayList<Prediction> predictions = gestureLibrary
                        .recognize(gesture);
                    ArrayList<String> result = new ArrayList<String>();
                    // 遍历所有找到的 Prediction 对象
                    for (Prediction pred : predictions)
                    {
                        // 只有相似度大于 2.0 的手势才会被输出
                        if (pred.score > 2.0)
                        {
                            result.add("与手势【" + pred.name + "】相似度为"
                                + pred.score);
                        }
                    }
                    if (result.size() > 0)
                    {
                        ArrayAdapter<Object> adapter = new
                            ArrayAdapter<Object>(RecogniseGesture.this,
                            android.R.layout.simple_dropdown_item_1line
                            , result.toArray());
                        // 使用一个带 List 的对话框来显示所有匹配的手势
                        new AlertDialog.Builder(RecogniseGesture.this)
                            .setAdapter(adapter, null)
                            .setPositiveButton("确定", null).show();
                    }
                    else
                    {
                        Toast.makeText(RecogniseGesture.this
                            , "无法找到能匹配的手势！",
                            Toast.LENGTH_LONG).show();
                    }
                }
            });
    }
}
```



图 8.15　识别手势

上面的程序中粗体字代码就负责调用前一个程序的手势库来识别用户刚输入的手势，用户只要在屏幕上绘一个大致与之前相似的手势，即可看到如图 8.15 所示的结果。



该程序需要读取 SD 卡上的手势文件，因此该程序还需要授予读取 SD 卡的权限。



8.5 自动朗读（TTS）

Android 提供了自动朗读支持。自动朗读支持可以对指定文本内容进行朗读，从而发生声音；不仅如此，Android 的自动朗读支持还允许把文本对应的音频录制成音频文件，方便以后播放。这种自动朗读支持的英文名称为 TextToSpeech，简称 TTS。借助于 TTS 的支持，可以在应用程序中动态地增加音频输出，从而改善用户体验。

Android 的自动朗读支持主要通过 TextToSpeech 来完成，该类提供了如下一个构造器：

- TextToSpeech(Context context, TextToSpeech.OnInitListener listener)

从上面的构造器不难看出，当创建 TextToSpeech 对象时，必须先提供一个 OnInitListener 监听器——该监听器负责监听 TextToSpeech 的初始化结果。

一旦在程序中获得了 TextToSpeech 对象之后，接下来可调用 TextToSpeech 的 setLanguage(Locale loc)方法来设置该 TTS 发声引擎应使用的语言、国家选项。



提示：

由于不同的文字，在不同的语言、国家中的发音是不同的，尤其是欧美，他们所使用的都是字母文字，因此一段文本内容，使用不同的语言、国家选项来朗读，发音效果是截然不同的。目前 Android 的 TTS 暂时不支持中文。

如果调用 setLanguage(Locale loc)的返回值是“TextToSpeech.LANG_COUNTRY_AVAILABLE”说明当前 TTS 系统可以支持所设置的语言、国家选项。

对 TextToSpeech 设置完成后，就可调用它的方法来朗读文本了，具体方法可参考 TextToSpeech 的 API 文档。TextToSpeech 类中最常用的方法是如下两个。

- speak(String text, int queueMode, HashMap<String, String> params)
- synthesizeToFile(String text, HashMap<String, String> params, String filename)

上面两个方法都用于把 text 文字内容转换为音频，区别只是 speak 方法是播放转换的音频，而 synthesizeToFile 是把转换得到的音频保存成声音文件。

上面两个方法中的 params 都用于指定声音转换时的参数，speak()方法中的 queueMode 参数指定 TTS 的发音队列模式，该参数支持如下两个常量。

- TextToSpeech.QUEUE_FLUSH：如果指定该模式，当 TTS 调用 speak 方法时，它会中断当前实例正在运行的任务（也可以理解为清除当前语音任务，转而执行新的语音任务）
- TextToSpeech.QUEUE_ADD：如果指定为该模式，当 TTS 调用 speak 方法时，会把新的发音任务添加到当前发音任务列队之后——也就是等任务队列中的发音任务执行完成后再来执行 speak()方法指定的发音任务。

当程序用完了 TextToSpeech 对象之后，可以在 Activity 的 OnDestroy()方法中调用它的 shutdown()来关闭 TextToSpeech、释放它所占用的资源。

归纳起来，使用 TextToSpeech 的步骤如下：

① 创建 TextToSpeech 对象，创建时传入 OnInitListener 监听器监听创建是否成功。

② 设置 TextToSpeech 所使用语言、国家选项，通过返回值判断 TTS 是否支持该语言、国家选项。

③ 调用 speak()或 synthesizeToFile 方法。

④ 关闭 TTS，回收资源。

下面的程序示范了如何利用 TTS 来朗读用户所输入的文本内容。

程序清单：codes\08\8.5\Speech\src\org\crazyit\io\Speech.java

```
public class Speech extends Activity
{
    TextToSpeech tts;
    EditText editText;
    Button speech;
    Button record;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 初始化 TextToSpeech 对象
        tts = new TextToSpeech(this, new OnInitListener()
        {
            @Override
            public void onInit(int status)
            {
                // 如果装载 TTS 引擎成功
                if (status == TextToSpeech.SUCCESS)
                {
                    // 设置使用美式英语朗读
                    int result = tts.setLanguage(Locale.US);
                    // 如果不支持所设置的语言
                    if (result != TextToSpeech.LANG_COUNTRY_AVAILABLE
                        && result != TextToSpeech.LANG_AVAILABLE)
                    {
                        Toast.makeText(Speech.this
                            , "TTS 暂时不支持这种语言的朗读。", Toast.LENGTH_
                            LONG)
                            .show();
                    }
                }
            }
        });
        editText = (EditText) findViewById(R.id.txt);
        speech = (Button) findViewById(R.id.speech);
        record = (Button) findViewById(R.id.record);
        speech.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 执行朗读
                tts.speak(editText.getText().toString(),
                    TextToSpeech.QUEUE_ADD, null);
            }
        });
        record.setOnClickListener(new OnClickListener()
        {
```

```
                @Override
                public void onClick(View arg0)
                {
                    // 将朗读文本的音频记录到指定文件
                    tts.synthesizeToFile(editText.getText().toString()
                        , null, "/mnt/sdcard/sound.wav");
                    Toast.makeText(Speech.this, "声音记录成功！"
                        , Toast.LENGTH_LONG).show();
                }
            });
        }
        @Override
        public void onDestroy()
        {
            // 关闭 TextToSpeech 对象
            if (tts != null)
            {
                tts.shutdown();
            }
        }
    }
```

上面的程序中第一行粗体字代码设置创建了一个 TextToSpeech 对象，第二行粗体字代码设置使用美式英语进行朗读。接下来程序分别提供了两个按钮，一个按钮用于执行朗读发生，一个按钮用于将文本内容朗读音频保存成声音文件，分别通过调用 TextToSpeech 对象的 speek()和 synthesizeToFile()方法完成——如上面的程序中后两行粗体字代码所示。



图 8.16 TTS

运行上面的程序，将可以看到 8.16 所示的界面。

在图 8.16 所示界面中，当用户单击“朗读”按钮后，系统将会调用 TTS 的 speek()方法来朗读文本框中的内容；当用户单击“记录声音”按钮后，系统将会调用 synthesizeToFile()方法把文本框中的文本对应的朗读音频记录到SD卡的声音文件中——单击该按钮后将可以在SD卡的根目录下生成一个 sound.wav 文件，该文件可以被导出，在其他音频播放软件中播放。

程序重写 Activity 的 onDestroy()方法，并在该方法中关闭了 TextToSpeech 对象，回收了它的资源。

8.6 本章小结

本章主要介绍了 Android 的输入、输出支持，如果读者之前已有了 Java IO 的编程经验，那么可以直接把 Java IO 的编程应验移植到 Android 上。当然 Android 为文件 IO 提供了 openFileOutput 和 openFileInput 两个便捷的方法，用起来十分方便；为记录、访问应用程序的参数、选项提供了 SharedPreferences 工具类，这样可以非常方便地读、写应用程序的参数、选项。除此之外，学习本章需要重点掌握的内容就是 SQLite 数据库，Android 系统内置了 SQLite 数据库，而且为访问 SQLite 数据库提供了大量方便的工具类，这需要读者掌握并能熟练地使用它们。

本章后面还介绍了 Android 提供的“另类”IO：手势支持和自动朗读。Android 的手势支持体现在两方面：手势检测与手势识别，前者属于事件处理方面，后者属于系统 IO 方面。熟练运用 Android 系统的手势支持可以开发出一些更新奇的应用。自动朗读则用于把文本转换成声音。

CHAPTER

9

第9章 使用ContentProvider实现数据共享

本章要点

- 理解 ContentProvider 的功能与意义
- ContentProvider 类的作用和常用方法
- Uri 对 ContentProvider 的作用
- 理解 ContentProvider 与 ContentResolver 的关系
- 实现自己的 ContentProvider
- 配置 ContentProvider
- 使用 ContentResolver 操作数据
- 操作系统 ContentProvider 提供的数据
- 监听 ContentProvider 的数据改变
- ContentObserver 类的作用和常用方法
- 监听系统 ContentProvider 的数据改变

当在系统中部署一个又一个 Android 应用之后，系统里将会包含多个 Android 应用，有时候就需要在不同的应用之间共享数据，比如现在有一个短信接收应用，用户想把接收到的陌生短信的发信人添加到联系人管理应用中，就需要在不同应用之间共享数据。对于这种需要在不同应用之间共享数据的需求，当然可以让一个应用程序直接去操作另一个应用程序所记录的数据，比如操作它所记录的 SharedPreferences、文件或数据库等，这种方式显得太杂乱了：不同的应用程序记录数据的方式差别很大，这种方式不利于应用程序之间进行数据交换。

为了在应用程序之间交换数据，Android 提供了 ContentProvider，ContentProvider 是不同应用程序之间进行数据交换的标准 API，当一个应用程序需要把自己的数据暴露给其他程序使用时，该应用程序就可通过提供 ContentProvider 来实现；其他应用程序就可通过 ContentResolver 来操作 ContentProvider 暴露的数据。

ContentProvider 也是 Android 应用的四大组件之一，与 Activity、Service、BroadcastReceiver 的相似，它们都需要在 AndroidManifest.xml 文件中进行配置。

一旦某个应用程序通过 ContentProvider 暴露了自己的数据操作接口，那么不管该应用程序是否启动，其他应用程序都可通过该接口来操作该应用程序的内部数据，包括增加数据、删除数据、修改数据、查询数据等。

9.1 数据共享标准：ContentProvider 简介

ContentProvider 是不同应用程序之间进行数据交换的标准 API，ContentProvider 以某种 Uri 的形式对外提供数据，允许其他应用访问或修改数据；其他应用程序使用 ContentResolver 根据 Uri 去访问操作指定数据。



提示：

对于初学者而言，对于 ContentProvider、ContentResolver 两个核心 API 的作用可能需要花很长时间去理解。但这里有一个简单的类比：可以把 ContentProvider 当成 Android 系统内部的“网站”，这个网站以固定的 Uri 对外提供服务；而 ContentResolver 则可当成 Android 系统内部的 HttpClient，它可以向指定 Uri 发送“请求”（实际上是调用 ContentResolver 的方法），这种请求最后委托给 ContentProvider 处理，从而实现对“网站”（即 ContentProvider）内部数据进行操作。

9.1.1 ContentProvider 简介

如果把 ContentProvider 当成一个“网站”来看，那么如何对外提供数据呢？是否需要像 Java Web 开发一样编写 JSP、Servlet 之类呢？不需要。如果那样就太复杂了，毕竟 ContentProvider 只是提供数据的访问接口，并不是像一个网站一样对外提供完整的页面。

如果把 ContentProvider 当成一个网站来看，那么如何完整地开发一个 ContentProvider 呢？步骤其实很简单：

① 定义自己的 ContentProvider 类，该类需要继承 Android 提供的 ContentProvider 基类。

② 向 Android 系统注册这个“网站”，也就是在 AndroidManifest.xml 文件中注册这个 ContentProvider，就像注册 Activity 一样。注册 ContentProvider 时需要为它绑定一个 Uri。

向 Android 系统中注册 ContentProvider 只要在<application.../>元素下添加如下子元素即

可：

```
<!-- 下面配置中 name 属性指定 ContentProvider 类
     authorities 就相当于为该 ContentProvider 指定域名 -->
<provider android:name=".DictProvider"
     android:authorities="org.crazyit.providers.dictprovider"
     android:exported="true"/>
```



提示：

虽然我们可以把 ContentProvider 当成一个网站来看，但 Android 官方文档并没有这种提法。Android 要求注册 ContentProvider 时指定 authorities 属性，该属性的值就相当于该网站的域名。但需要提醒读者：面试时千万不要这么说，因为你的面试官可能由于自身技术层次看不到这个高度，而并不认同这个说法，从而导致面试失败。

当我们通过上面配置文件注册了上面的 DictProvider 之后，其他应用程序就可通过该 Uri 来访问 DictProvider 所暴露的数据了。

那么 DictProvider 到底如何暴露它所提供的数据呢？其实很简单，应用程序对数据的操作无非就是 CRUD 操作，因此 DictProvider 除了需要继承 ContentProvider 之外，还需要提供如下几个方法。

- public boolean onCreate()：该方法在 ContentProvider 创建后会被调用，当其他应用程序第一次访问 ContentProvider 时，该 ContentProvider 会被创建出来，并立即回调该 onCreate()方法。
- public Uri insert(Uri uri, ContentValues values)：根据该 Uri 插入 values 对应的数据。
- public int delete(Uri uri, String selection, String[] selectionArgs)：根据 Uri 删除 select 条件所匹配的全部记录。
- public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)：根据 Uri 修改 select 条件所匹配的全部记录。
- public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：根据 Uri 查询出 select 条件所匹配的全部记录，其中 projection 就是一个列名列表，表明只选择出指定的数据列。
- public String getType(Uri uri)：该方法用于返回当前 Uri 所代表的数据的 MIME 类型。如果该 Uri 对应数据可能包括多条记录，那么 MIME 类型字符串应该以 vnd.android.cursor.dir/开头；如果该 Uri 对应的数据只包含一条记录，那么返回 MIME 类型字符串应该以 vnd.android.cursor.item/开头。

通过介绍不难发现，对于 ContentProvider 而言，Uri 是一个非常重要的概念，下面详细介绍 Uri 的相关知识。

9.1.2 Uri 简介

在介绍 Android 系统的 Uri 之前，先来看一个最常用的互联网 URL，例如想访问疯狂 Java 联盟的某个页面，应该在浏览器中输入如下 Uri：

http://www.crazyit.org:80/ethos.php

对于上面这个 URL，可分为如下三个部分。

- http://：URL 的协议部分，只要通过 HTTP 协议来访问网站，这个部分是固定的。
- www.crazyit.org：域名部分。只要访问指定的网站，这个部分总是固定的。
- ethos.php：网站资源部分。当访问者需要访问不同资源时，这个部分是动态改变的。

ContentProvider 要求的 Uri 与此类似，例如如下 Uri：

```
content://org.crazyit.providers.dictprovider/words
```

它也可分为如下三个部分。

- content://：这个部分是 Android 的 ContentProvider 规定的，就像上网的协议默认是 http://一样。暴露 ContentProvider、访问 ContentProvider 的协议默认是 content://。
- org.crazyit.providers.dictprovider：这个部分就是 ContentProvider 的 authority。系统就是由这个部分来找到操作哪个 ContentProvider。只要访问指定的 ContentProvider，这个部分总是固定的。
- words：资源部分（或者说数据部分），当访问者需要访问不同资源时，这个部分是动态改变的。

需要指出的是，Android 的 Uri 所能表达的功能更丰富，它还可以支持如下 Uri：

```
content://org.crazyit.providers.dictprovider/word/2
```

此时它要访问的资源为 word/2，这意味着访问 word 数据中 ID 为 2 的记录。

还有如下形式：

```
content://org.crazyit.providers.dictprovider/word/2/word
```

此时它要访问的资源为 word/2，这意味着访问 word 数据中 ID 为 2 的记录的 word 字段。

如果想访问全部数据，即可使用下面所示的形式：

```
content://org.crazyit.providers.dictprovider/words
```

虽然大部分使用 ContentProvider 所操作的数据都来自于数据库，但有时候这些数据也可来自于文件、XML 或网络等其他存储方式，此时支持的 Uri 也可以改为如下形式：

```
content://org.crazyit.providers.dictprovider/word/detail/
```

上面的 Uri 表示操作 word 节点下的 detail 节点。

为了将一个字符串转换成 Uri，Uri 工具类提供了 parse()静态方法，例如如下代码即可将字符串转换为 Uri：

```
Uri uri = Uri.parse("content://org.crazyit.providers.dictprovider/word/2")
```

9.1.3 使用 ContentResolver 操作数据

前面已经提到，ContentProvider 相当于一个“网站”，它的作用是暴露可供操作的数据；其他应用程序则通过 ContentResolver 来操作 ContentProvider 所暴露的数据，ContentResolver 相当于 HttpClient。

Context 提供了如下方法来获取 ContentResolver 对象。

- getContentResolver()

一旦在程序中获得 ContentResolver 对象之后，接下来就可调用 ContentResolver 的如下方

法来操作数据。

- insert(Uri url, ContentValues values)：向 Uri 对应的 ContentProvider 中插入 values 对应的数据。
- delete(Uri url, String where, String[] selectionArgs)：删除 Uri 对应的 ContentProvider 中 where 提交匹配的数据。
- update(Uri uri, ContentValues values, String where, String[] selectionArgs)：更新 Uri 对应的 ContentProvider 中 where 提交匹配的数据。
- query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)：查询 Uri 对应的 ContentProvider 中 where 提交匹配的数据。

一般来说，ContentProvider 是单实例模式的，当多个应用程序通过 ContentResolver 来操作 ContentProvider 提供的数据时，ContentResolver 调用的数据操作将会委托给同一个 ContentProvider 处理。

9.2 开发 ContentProvider

对于许多初学者而言，理解 ContentProvider 暴露数据的方式是一个难点。ContentProvider 总是需要与 ContentResolver 结合使用，ContentProvider 负责暴露数据，因此 ContentProvider 需要与 ContentResolver 结合学习。

9.2.1 ContentProvider 与 ContentResolver 的关系

从 ContentResolver、ContentProvider 和 Uri 的关系来看，无论是 ContentResolver，还是 ContentProvider，它们所提供的 CRUD 方法的第一个参数都是 Uri。也就是说，Uri 是 ContentResolver 和 ContentProvider 进行数据交换的标识。ContentResolver 对指定 Uri 执行 CRUD 等数据操作，但 Uri 并不是真正的数据中心，因此这些 CRUD 操作会委托给该 Uri 对应的 ContentProvider 来实现。通常来说，假如 A 应用通过 ContentResolver 执行 CRUD 操作，这些 CRUD 操作都需要指定 Uri 参数，Android 系统就根据该 Uri 找到对应的 ContentProvider（该 ContentProvider 通常属于 B 应用），ContentProvider 则负责实现 CRUD 方法，完成对底层数据的增、删、改、查等操作，这样就可以让 A 应用访问、修改 B 应用的数据。

ContentResolver、Uri、ContentProvider 三者之间的关系如图 9.1 所示。

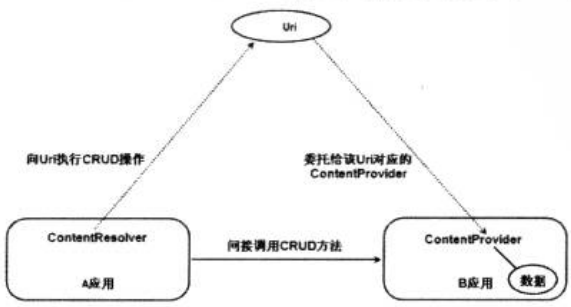


图 9.1　ContentRosolver 与 ContentProvider 的关系

从图 9.1 可以看出，以指定 Uri 为标识，ContentResolver 可以实现“间接调用”ContentProvider 的 CRUD 方法：

- 当 A 应用调用 ContentResolver 的 insert()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 insert()方法。
- 当 A 应用调用 ContentResolver 的 update()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 update()方法。
- 当 A 应用调用 ContentResolver 的 delete ()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 delete()方法。
- 当 A 应用调用 ContentResolver 的 query ()方法时，实际上相当于调用了该 Uri 对应的 ContentProvider（该 ContentProvider 属于 B 应用）的 query()方法。

通过上面这种关系，即可实现让 A 应用访问、使用 B 应用底层的数据。

9.2.2 开发 ContentProvider

开发 ContentProvider 只要如下两步：

① 开发一个 ContentProvider 的子类，该子类需要实现 query()、insert()、update()和 delete()等方法。

② 在 AndroidManifest.xml 文件中注册该 ContentProvider，指定 android:authorities 属性。

在上面两步中，ContentProvider 子类实现的 query()、insert()、update()和 delete()方法，并不是给该应用本身调用的，而是供其他应用来调用的。正如前面提到的，当其他应用通过 ContentResolver 调用 query()、insert()、update()和 delete()方法执行数据访问时，实际上就是调用指定 Uri 对应的 ContentProvider 的 query()、insert()、update()和 delete()方法。

如何实现 ContentProvider 的 query()、insert()、update()和 delete()方法，完全由程序员决定——程序员想怎么暴露该应用数据，就怎么实现这 4 个方法。极端情况下，开发者只对这些方法提供空实现也是可以的。

例如下面的示例 ContentProvider，该 ContentProvider 虽然实现了 query()、insert()、update()和 delete()方法，但并未真正对底层数据进行访问，只是输出了一行字符串。下面是该 ContentProvider 的代码。

程序清单：codes\09\9.2\FirstProvider\src\org\crazyit\content\FirstProvider.java

```
public class FirstProvider extends ContentProvider
{
    // 第一次创建该 ContentProvider 时调用该方法
    @Override
    public boolean onCreate()
    {
        System.out.println("===onCreate 方法被调用===");
        return true;
    }
    // 该方法的返回值代表了该 ContentProvider 所提供数据的 MIME 类型
    @Override
    public String getType(Uri uri)
    {
        return null;
    }
    // 实现查询方法，该方法应该返回查询得到的 Cursor
    @Override
    public Cursor query(Uri uri, String[] projection, String where,
        String[] whereArgs, String sortOrder)
    {
        System.out.println(uri + "===query 方法被调用===");
```

```
            System.out.println("where 参数为：" + where);
            return null;
        }
        // 实现插入的方法，该方法应该返回新插入的记录的 Uri
        @Override
        public Uri insert(Uri uri, ContentValues values)
        {
            System.out.println(uri + "===insert 方法被调用===");
            System.out.println("values 参数为：" + values);
            return null;
        }
        // 实现删除方法，该方法应该返回被更新的记录条数
        @Override
        public int update(Uri uri, ContentValues values, String where,
            String[] whereArgs)
        {
            System.out.println(uri + "===update 方法被调用===");
            System.out.println("where 参数为："
                + where + ",values 参数为：" + values);
            return 0;
        }
        // 实现删除方法，该方法应该返回被删除的记录条数
        @Override
        public int delete(Uri uri, String where, String[] whereArgs)
        {
            System.out.println(uri + "===delete 方法被调用===");
            System.out.println("where 参数为：" + where);
            return 0;
        }
}
```

上面 4 个粗体字方法实现了 query()、insert()、update()和 delete()方法，这 4 个方法用于供其他应用通过 ContentRosovler 调用。

在该 ContentProvider 供其他应用调用之前，还需要先配置该 ContentProvider。

9.2.3 配置 ContentProvider

Android 应用要求所有应用程序组件（Activity、Service、ContentProvider、BroadcastReceiver）都必须显式进行配置。

只要为<application.../>元素添加<provider.../>子元素即可配置 ContentProvider。例如如下的配置片段：

```
<provider
    android:name=".FirstProvider"
    android:authorities="org.crazyit.providers.firstprovider"
    android:exported="true" />
```

从上面的配置片段可以看出，配置 ContentProvider 时通常指定如下属性。

- name：指定该 ContentProvider 的实现类的类名。
- authorities：指定该 ContentProvider 对应的 Uri（相当于为该 ContentProvider 分配一个域名）。
- android:exported：指定该 ContentProvider 是否允许其他应用调用。如果将该属性设为 false，那么该 ContentProvider 将不允许其他应用调用。

为了配置上面的 ContentProvider，只要在<application.../>元素中添加如下子元素即可。

程序清单：codes\09\9.2\FirstProvider\AndroidManifest.xml

```
<application
      android:icon="@drawable/ic_launcher">
      <!-- 注册一个 ContentProvider -->
      <provider
            android:name=".FirstProvider"
            android:authorities="org.crazyit.providers.firstprovider"
            android:exported="true" />
</application>
```

上面的配置指定了该 ContentProvider 被绑定到“content://org.crazyit.providers.firstprovider”——这意味着，当其他应用的 ContentResovler 向该 Uri 执行 query()、insert()、update()和 delete()方法时，实际上是调用该 ContentProvider 的 query()、insert()、update()和 delete()方法。

- ContentResovler 调用方法时参数将会传给该 ContentProvider 的 query()、insert()、update()和 delete()方法。
- ContentResovler 调用方法的返回值，也就是 ContentProvider 执行 query()、insert()、update()和 delete()方法的返回值。

9.2.4 使用 ContentResolver 调用方法

Context 提供了 getContentResovler()方法，这表明 Activity、Service 等组件都可通过 getContentResovler()方法获取 ContentResolver 对象。

获取了 ContentResovler 对象之后，接下来就可调用 ContentResolver 的 query()、insert()、update()和 delete()方法了——实际上是指定 Uri 对应的 ContentProvider 的 query()、insert()、update()和 delete()。

下面示例的界面文件中只包含了 4 个按钮，4 个按钮分别激发调用 ContentProvider 的 query()、insert()、update()和 delete()方法，由于该示例的界面布局文件很简单，故此处不再给出。

下面是该示例的 Activity 代码。

程序清单：codes\09\9.2\FirstResolver\src\org\crazyit\resolver\FirstResolver.java

```
public class FirstResolver extends Activity
{
      ContentResolver contentResolver;
      Uri uri = Uri.parse("content://org.crazyit.providers.firstprovider/");
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取系统的 ContentResolver 对象
            contentResolver = getContentResolver();
      }
      public void query(View source)
      {
            // 调用 ContentResolver 的 query()方法
            // 实际返回的是该 Uri 对应的 ContentProvider 的 query()的返回值
            Cursor c = contentResolver.query(uri, null
                  , "query_where", null, null);
            Toast.makeText(this, "远程 ContentProvide 返回的 Cursor 为：" + c,
```

```
                Toast.LENGTH_LONG).show();
        }
        public void insert(View source)
        {
            ContentValues values = new ContentValues();
            values.put("name", "fkjava");
            // 调用 ContentResolver 的 insert()方法。
            // 实际返回的是该 Uri 对应的 ContentProvider 的 insert()的返回值
            Uri newUri = contentResolver.insert(uri, values);
            Toast.makeText(this, "远程 ContentProvide 新插入记录的 Uri 为："
                + newUri, Toast.LENGTH_LONG).show();
        }
        public void update(View source)
        {
            ContentValues values = new ContentValues();
            values.put("name", "fkjava");
            // 调用 ContentResolver 的 update()方法。
            // 实际返回的是该 Uri 对应的 ContentProvider 的 update()的返回值
            int count = contentResolver.update(uri, values
                , "update_where", null);
            Toast.makeText(this, "远程 ContentProvide 更新记录数为："
                + count, Toast.LENGTH_LONG).show();
        }
        public void delete(View source)
        {
            // 调用 ContentResolver 的 delete()方法
            // 实际返回的是该 Uri 对应的 ContentProvider 的 delete()的返回值
            int count = contentResolver.delete(uri
                , "delete_where", null);
            Toast.makeText(this, "远程 ContentProvide 删除记录数为："
                + count, Toast.LENGTH_LONG).show();
        }
    }
```

上面的 4 行粗体字代码通过 ContentResolver 调用 query()、insert()、update()和 delete()方法，实际上就是调用 uri 参数对应的 ContentProvider 的 query()、insert()、update()和 delete()方法，也就是前面的 FirstProvider 的 query()、insert()、update()和 delete()方法。

运行上面的程序，并依次单击该应用的 4 个按钮，将可以看到 LogCat 生成如图 9.2 所示的输出。

从图 9.2 所示的输出可以看出，当 Android 应用通过 ContentResolver 调用 query()、insert()、update()和 delete()方法时，实际上就是调用 FirstProvider（该 Uri 对应的 ContentProvider）的 query()、insert()、update()和 delete()方法。



图 9.2　ContentProvider 的方法被调用

当用户单击程序界面的“插入”按钮时，可以在程序界面看到如图 9.3 所示的输出。

从图 9.3 可以看出，调用 ContentResovler 的 insert()方法的返回值为 null，这是因为 FirstProvider 实现 insert()方法时返回了 null。

9.2.5 创建 ContentProvider 的说明



图 9.3 通过 ContentResovler 调用远程 ContentProvider 的方法

通过上面的介绍可以看出：ContentProvider 不像 Activity 存在复杂的生命周期，ContentProvider 只有一个 onCreate() 生命周期方法——当其他应用通过 ContentResolver 第一次访问该 ContentProvider 时，onCreate()方法将会被回调，onCreate()方法只会被调用一次；ContentProvider 提供的 query()、insert()、update()和 delete()方法则由其他应用通过 ContentResolver 调用。

开发 ContentProvider 时所实现的 query()、insert()、update()和 delete()方法的第一个参数为 Uri 参数，该参数由 ContentResolver 调用这些方法时传入。

前面介绍的示例由于并未真正对数据进行操作，因此 ContenProvider 并未对 Uri 参数进行任何判断。实际上，为了确定该 ContentProvider 实际能处理的 Uri，以及确定每个方法中 Uri 参数所操作的数据，Android 系统提供了 UriMatcher 工具类。

UriMatcher 工具类主要提供了如下两个方法。

- **void addURI(String authority, String path, int code)**：该方法用于向 UriMatcher 对象注册 Uri。其中 authority 和 path 组合成一个 Uri，而 code 则代表该 Uri 对应的标识码。
- **int match(Uri uri)**：根据前面注册的 Uri 来判断指定 Uri 对应的标识码。如果找不到匹配的标识码，该方法将会返回-1。

例如我们先通过如下代码来创建 UriMatcher 对象：

```
UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
matcher.addURI("org.crazyit.providers.dictprovider", "words", 1);
matcher.addURI("org.crazyit.providers.dictprovider", "word/#", 2);
```

上面创建的 UriMatcher 对象注册了两个 Uri，其中 org.crazyit.providers.dictprovider/words 对应的标识码为 1；org.crazyit.providers.dictprovider/word/#对应的标识码为 2，其中#为通配符。

这就意味着如下匹配结果：

```
matcher.match(Uri.parse("content://org.crazyit.providers.dictprovider/words"));
// 返回标识码 1
matcher.match(Uri.parse("content://org.crazyit.providers.dictprovider/word/2"));
// 返回标识码 2
matcher.match(Uri.parse("content://org.crazyit.providers.dictprovider/word/10"));
// 返回标识码 2
```

至于到底需要为 UriMatcher 对象注册多少个 Uri，取决于系统的业务需求。

对于 content://org.crazyit.providers.dictprovider/words 这个 Uri，它的资源部分为 words，这种资源通常代表了访问所有数据项；对于 content://org.crazyit.providers.dictprovider/word/2 这个 Uri，它的资源部分通常代表访问指定数据项，其中最后一个数值往往代表了该数据的 ID。

除此之外，Android 还提供了一个 ContentUris 工具类，它是一个操作 Uri 字符串的工具类，提供了如下两个工具方法。

- **withAppendedId(uri, id)**：用于为路径加上 ID 部分。例如：

```
Uri uri = Uri.parse("content://org.crazyit.providers.dictprovider/word")
```

```
Uri resultUri = ContentUris.withAppendedId(uri, 2);
// 生成后的 Uri 为："content://org.crazyit.providers.dictprovider/word/2"
```

- parseId(uri)：用于从指定 Uri 中解析出所包含的 ID 值。例如：

```
Uri uri = Uri.parse("content://org.crazyit.providers.dictprovider/word/2")
long wordId = ContentUris.parseId(uri); // 获取的结果为:2
```

掌握上面的知识之后，接下来我们将对前面介绍的生词本应用进行改进，增加 ContentProvider，这样即可允许其他应用程序通过 ContentResolver 来操作生词本应用的数据。

实例：使用 ContentProvider 共享生词本数据

正如前面访问系统 ContentProvider 所见到的，系统一般都会把 ContentProvider 的 Uri、数据列等信息以常量的形式公开出来，方便访问。为此，我们也为该 ContentProvider 定义一个工具类，该类中只是包含一个 public static 的常量。该工具类的代码如下。

程序清单：codes\09\9.2\DictProvider\src\org\crazyit\content\Words.java

```
public final class Words
{
    // 定义该 ContentProvider 的 Authority
    public static final String AUTHORITY
        = "org.crazyit.providers.dictprovider";

    // 定义一个静态内部类，定义该 ContentProvider 所包含的数据列的列名
    public static final class Word implements BaseColumns
    {
        // 定义 Content 所允许操作的三个数据列
        public final static String _ID = "_id";
        public final static String WORD = "word";
        public final static String DETAIL = "detail";
        // 定义该 Content 提供服务的两个 Uri
        public final static Uri DICT_CONTENT_URI = Uri
            .parse("content://" + AUTHORITY + "/words");
        public final static Uri WORD_CONTENT_URI = Uri
            .parse("content://"  + AUTHORITY + "/word");
    }
}
```

上面的工具类只是定义了一些简单的常量的工具类，这个工具类的作用就是告诉其他应用程序，访问该 ContentProvider 的一些常用入口。



提示：

实际上我们也可以不提供 Words 工具类，而是在 ContentProvider 中直接使用字符串来定义提供服务的 Uri。接下来再采用文档去告诉其他应用程序访问该 ContentProvider 的入口。但这种方式的可维护性并不好，因此实际项目中（包括 Android 系统的 ContentProvider）都会采用工具类来定义各种常量的方式进行处理。

本示例只是在前面介绍的生词本应用的基础上进行修改，因此本应用可以直接使用前面所开发的 SQLiteOpenHelper 类。

接下来我们开发一个 ContentProvider 的子类，并重写其中的增、删、改、查等方法，类代码如下。

程序清单：codes\09\9.2\DictProvider\src\org\crazyit\content\DictProvider.java

```
public class DictProvider extends ContentProvider
{
    private static UriMatcher matcher
        = new UriMatcher(UriMatcher.NO_MATCH);
    private static final int WORDS = 1;
    private static final int WORD = 2;
    private MyDatabaseHelper dbOpenHelper;
    static
    {
        // 为 UriMatcher 注册两个 Uri
        matcher.addURI(Words.AUTHORITY, "words", WORDS);
        matcher.addURI(Words.AUTHORITY, "word/#", WORD);
    }
    // 第一次调用该 DictProvider 时，系统先创建 DictProvider 对象，并回调该方法
    @Override
    public boolean onCreate()
    {
        dbOpenHelper = new MyDatabaseHelper(this.getContext(),
            "myDict.db3", 1);
        return true;
    }
    // 返回指定 Uri 参数对应的数据的 MIME 类型
    @Override
    public String getType(Uri uri)
    {
        switch (matcher.match(uri))
        {
            // 如果操作的数据是多项记录
            case WORDS:
                return "vnd.android.cursor.dir/org.crazyit.dict";
            // 如果操作的数据是单项记录
            case WORD:
                return "vnd.android.cursor.item/org.crazyit.dict";
            default:
                throw new IllegalArgumentException("未知 Uri:" + uri);
        }
    }
    // 查询数据的方法
    @Override
    public Cursor query(Uri uri, String[] projection, String where,
        String[] whereArgs, String sortOrder)
    {
        SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
        switch (matcher.match(uri))
        {
            // 如果 Uri 参数代表操作全部数据项
            case WORDS:
                // 执行查询
                return db.query("dict", projection, where,
                    whereArgs, null, null, sortOrder);
            // 如果 Uri 参数代表操作指定数据项
            case WORD:
                // 解析出想查询的记录 ID
                long id = ContentUris.parseId(uri);
                String whereClause = Words.Word._ID + "=" + id;
                // 如果原来的 where 子句存在，拼接 where 子句
                if (where != null && !"".equals(where))
                {
                    whereClause = whereClause + " and " + where;
```

```
                }
                return db.query("dict", projection, whereClause, whereArgs,
                        null, null, sortOrder);
            default:
                throw new IllegalArgumentException("未知Uri:" + uri);
        }
    }
    // 插入数据方法
    @Override
    public Uri insert(Uri uri, ContentValues values)
    {
        // 获得数据库实例
        SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
        switch (matcher.match(uri))
        {
            // 如果Uri参数代表操作全部数据项
            case WORDS:
                // 插入数据，返回插入记录的ID
                long rowId = db.insert("dict", Words.Word._ID, values);
                // 如果插入成功返回uri
                if (rowId > 0)
                {
                    // 在已有的 Uri 的后面追加 ID
                    Uri wordUri = ContentUris.withAppendedId(uri, rowId);
                    // 通知数据已经改变
                    getContext().getContentResolver().notifyChange(wordUri,
                    null);
                    return wordUri;
                }
                break;
            default :
                throw new IllegalArgumentException("未知Uri:" + uri);
        }
        return null;
    }
    // 修改数据的方法
    @Override
    public int update(Uri uri, ContentValues values, String where,
        String[] whereArgs)
    {
        SQLiteDatabase db = dbOpenHelper.getWritableDatabase();
        // 记录所修改的记录数
        int num = 0;
        switch (matcher.match(uri))
        {
            // 如果Uri参数代表操作全部数据项
            case WORDS:
                num = db.update("dict", values, where, whereArgs);
                break;
            // 如果Uri参数代表操作指定数据项
            case WORD:
                // 解析出想修改的记录ID
                long id = ContentUris.parseId(uri);
                String whereClause = Words.Word._ID + "=" + id;
                // 如果原来的where子句存在，拼接where子句
                if (where != null && !where.equals(""))
                {
                    whereClause = whereClause + " and " + where;
                }
                num = db.update("dict", values, whereClause, whereArgs);
                break;
```

```
                default:
                    throw new IllegalArgumentException("未知Uri:" + uri);
            }
            // 通知数据已经改变
            getContext().getContentResolver().notifyChange(uri, null);
            return num;
        }
        // 删除数据的方法
        @Override
        public int delete(Uri uri, String where, String[] whereArgs)
        {
            SQLiteDatabase db = dbOpenHelper.getReadableDatabase();
            // 记录所删除的记录数
            int num = 0;
            // 对uri进行匹配
            switch (matcher.match(uri))
            {
                // 如果Uri参数代表操作全部数据项
                case WORDS:
                    num = db.delete("dict", where, whereArgs);
                    break;
                // 如果Uri参数代表操作指定数据项
                case WORD:
                    // 解析出所需要删除的记录ID
                    long id = ContentUris.parseId(uri);
                    String whereClause = Words.Word._ID + "=" + id;
                    // 如果原来的where子句存在，拼接where子句
                    if (where != null && !where.equals(""))
                    {
                        whereClause = whereClause + " and " + where;
                    }
                    num = db.delete("dict", whereClause, whereArgs);
                    break;
                default:
                    throw new IllegalArgumentException("未知Uri:" + uri);
            }
            // 通知数据已经改变
            getContext().getContentResolver().notifyChange(uri, null);
            return num;
        }
    }
```

上面的DictProvider类很简单，它除了继承系统的ContentProvider之外，还实现了操作数据的增、删、改、查等方法，那么该ContentProvider就开发完成了。该DictProvider真正调用了SQLiteDatabase对底层数据执行增、删、改、查，当其他应用通过ContentResolver来调用该DictProvider的query()、insert()、update()和delete()方法时，就可以真正访问、操作该ContentProvider所在应用的底层数据。

接下来需要在AndroidManifest.xml文件中注册该ContentProvider，这就需要在AndroidManifest.xml文件中增加如下配置片段：

```
<!-- 注册一个ContentProvider -->
<provider android:name=".DictProvider"
    android:authorities="org.crazyit.providers.dictprovider"
    android:exported="true"/>
```

至此，暴露生词本数据的ContentProvider开发完成。为了测试该ContentProvider的开发是否成功，接下来再开发一个应用程序，该应用程序将会通过ContentResolver来操作生词本中的数据。

该程序同样提供了添加生词、查询生词的功能，只是该程序自己并不保存数据，而是访问前面 DictProvider 所共享的数据。下面是使用 ContentResolver 的类的代码。

程序清单：codes\09\9.2\DictResolver\src\org\crazyit\resolver\DictResolvertTest.java

```
public class DictResolverTest extends Activity
{
    ContentResolver contentResolver;
    Button insert = null;
    Button search = null;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 ContentResolver 对象
        contentResolver = getContentResolver();
        insert = (Button) findViewById(R.id.insert);
        search = (Button) findViewById(R.id.search);
        // 为 insert 按钮的单击事件绑定事件监听器
        insert.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String word = ((EditText) findViewById(R.id.word))
                    .getText().toString();
                String detail = ((EditText) findViewById(R.id.detail))
                    .getText().toString();
                // 插入生词记录
                ContentValues values = new ContentValues();
                values.put(Words.Word.WORD, word);
                values.put(Words.Word.DETAIL, detail);
                contentResolver.insert(
                    Words.Word.DICT_CONTENT_URI, values);
                // 显示提示信息
                Toast.makeText(DictResolverTest.this, "添加生词成功！"
                    , Toast.LENGTH_LONG).show();
            }
        });
        // 为 search 按钮的单击事件绑定事件监听器
        search.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入
                String key = ((EditText) findViewById(R.id.key))
                    .getText().toString();
                // 执行查询
                Cursor cursor = contentResolver.query(
                    Words.Word.DICT_CONTENT_URI, null,
                    "word like ? or detail like ?", new String[] {
                        "%" + key + "%", "%" + key + "%" }, null);
                // 创建一个 Bundle 对象
                Bundle data = new Bundle();
                data.putSerializable("data", converCursorToList(cursor));
                // 创建一个 Intent
                Intent intent = new Intent(DictResolverTest.this,
                    ResultActivity.class);
```

```
                intent.putExtras(data);
                // 启动Activity
                startActivity(intent);
            }
        });
    }
    private ArrayList<Map<String, String>> converCursorToList(Cursor cursor)
    {
        ArrayList<Map<String, String>> result
            = new ArrayList<Map<String, String>>();
        // 遍历Cursor结果集
        while (cursor.moveToNext())
        {
            // 将结果集中的数据存入ArrayList中
            Map<String, String> map = new HashMap<String, String>();
            // 取出查询记录中第2列、第3列的值
            map.put(Words.Word.WORD, cursor.getString(1));
            map.put(Words.Word.DETAIL, cursor.getString(2));
            result.add(map);
        }
        return result;
    }
}
```

上面的程序中第一行粗体字代码用于向 DictProvider 所共享的数据中添加记录；第二行粗体字代码则用于查询 DictProvider 所共享的数据。

运行该程序需要先部署前面介绍的 DictProvider，因为该程序所操作的数据实际上来自于 DictProvider。当用户通过该程序来添加生词时，实际上是添加到了前面所介绍的 DictProvider 应用中；当用户通过该程序来查询生词时，实际上是查询前面介绍的 DictProvider 应用中的生词。

9.3 操作系统的 ContentProvider

实际上，Android 系统本身提供了大量的 ContentProvider，例如联系人信息、系统的多媒体信息等，开发者自己开发的 Android 应用也可通过 ContentResolver 来调用系统 ContentProvider 提供的 query()、insert()、update()和 delete()方法，这样开发者即可通过这些 ContentProvider 来获取 Android 内部的数据。

使用 ContentResolver 操作操作系统 ContentProvider 数据的步骤依然是两步：

① 调用 Activity 的 getContentResolver()获取 ContentResolver 对象。

② 根据需要调用 ContentResolver 的 insert()、delete()、update()和 query 方法操作数据即可。

为了操作系统提供的 ContentResolver，需要了解该 ContentProvider 的 Uri，以及该 ContentProvider 所操作的数据列的列名，可以通过查阅 Android 官方文档来获取这些信息。

9.3.1 使用 ContentProvider 管理联系人

Android 系统提供了 Contacts 应用程序来管理联系人，而且 Android 系统还为联系人管理提供 ContentProvider，这就允许其他应用程序以 ContentResolver 来管理联系人数据。

Android 系统对联系人管理 ContentProvider 的几个 Uri 如下。

- ContactsContract.Contacts.CONTENT_URI：管理联系人的 Uri。
- ContactsContract.CommonDataKinds.Phone.CONTENT_URI：管理联系人的电话的 Uri。
- ContactsContract.CommonDataKinds.Email.CONTENT_URI：管理联系人的 E-mail 的 Uri。

了解联系人管理 ContentProvider 的 Uri 之后，接下来就可在应用程序中通过 ContentResolver 去操作系统的联系人数据了。

下面示例程序中包含两个按钮，一个按钮用于查询系统的联系人数据，该按钮所绑定的事件监听器代码如下。

程序清单：codes\09\9.3\ContactProviderTest\src\org\crazyit\content\ContactProviderTest.java

```
search.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View source)
    {
        // 定义两个 List 来封装系统的联系人信息、指定联系人的电话号码、Email 等详情
        final ArrayList<String> names = new ArrayList<String>();
        final ArrayList<ArrayList<String>> details
            = new ArrayList<ArrayList<String>>();
        // 使用 ContentResolver 查找联系人数据
        Cursor cursor = getContentResolver().query(
            ContactsContract.Contacts.CONTENT_URI, null, null,
            null, null);
        // 遍历查询结果，获取系统中所有联系人
        while (cursor.moveToNext())
        {
            // 获取联系人 ID
            String contactId = cursor.getString(cursor
                .getColumnIndex(ContactsContract.Contacts._ID));
            // 获取联系人的名字
            String name = cursor.getString(cursor.getColumnIndex(
                ContactsContract.Contacts.DISPLAY_NAME));
            names.add(name);
            // 使用 ContentResolver 查找联系人的电话号码
            Cursor phones = getContentResolver().query(
                ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                null,
                ContactsContract.CommonDataKinds.Phone.CONTACT_ID
                    + " = " + contactId, null, null);
            ArrayList<String> detail = new ArrayList<String>();
            // 遍历查询结果，获取该联系人的多个电话号码
            while (phones.moveToNext())
            {
                // 获取查询结果中电话号码列中数据
                String phoneNumber = phones.getString(phones
                    .getColumnIndex(ContactsContract
                    .CommonDataKinds.Phone.NUMBER));
                detail.add("电话号码：" + phoneNumber);
            }
            phones.close();
            // 使用 ContentResolver 查找联系人的 E-mail 地址
            Cursor emails = getContentResolver().query(
                ContactsContract.CommonDataKinds.Email.CONTENT_URI,
                null,
                ContactsContract.CommonDataKinds.Email.CONTACT_ID
                + " = " + contactId, null, null);
```

```
        // 遍历查询结果，获取该联系人的多个 E-mail 地址
        while (emails.moveToNext())
        {
            // 获取查询结果中 E-mail 地址列中数据
            String emailAddress = emails.getString(emails
                .getColumnIndex(ContactsContract
                .CommonDataKinds.Email.DATA));
            detail.add("邮件地址：" + emailAddress);
        }
        emails.close();
        details.add(detail);
    }
    cursor.close();
    // 加载 result.xml 界面布局代表的视图
    View resultDialog = getLayoutInflater().inflate(
        R.layout.result, null);
    // 获取 resultDialog 中 ID 为 list 的 ExpandableListView
    ExpandableListView list = (ExpandableListView) resultDialog
        .findViewById(R.id.list);
    // 创建一个 ExpandableListAdapter 对象
    ExpandableListAdapter adapter =
        new BaseExpandableListAdapter()
    {
        // 获取指定组位置、指定子列表项处的子列表项数据
        @Override
        public Object getChild(int groupPosition,
            int childPosition)
        {
            return details.get(groupPosition).get(
                childPosition);
        }
        @Override
        public long getChildId(int groupPosition,
            int childPosition)
        {
            return childPosition;
        }
        @Override
        public int getChildrenCount(int groupPosition)
        {
            return details.get(groupPosition).size();
        }
        private TextView getTextView()
        {
            AbsListView.LayoutParams lp = new AbsListView
                .LayoutParams(ViewGroup.LayoutParams.MATCH_PARENT
                , 64);
            TextView textView = new TextView(
                ContactProviderTest.this);
            textView.setLayoutParams(lp);
            textView.setGravity(Gravity.CENTER_VERTICAL
                | Gravity.LEFT);
            textView.setPadding(36, 0, 0, 0);
            textView.setTextSize(20);
            return textView;
        }
        // 该方法决定每个子选项的外观
        @Override
        public View getChildView(int groupPosition,
            int childPosition, boolean isLastChild,
            View convertView, ViewGroup parent)
```

```
                {
                    TextView textView = getTextView();
                    textView.setText(getChild(groupPosition,
                        childPosition).toString());
                    return textView;
                }
                // 获取指定组位置处的组数据
                @Override
                public Object getGroup(int groupPosition)
                {
                    return names.get(groupPosition);
                }
                @Override
                public int getGroupCount()
                {
                    return names.size();
                }
                @Override
                public long getGroupId(int groupPosition)
                {
                    return groupPosition;
                }
                // 该方法决定每个组选项的外观
                @Override
                public View getGroupView(int groupPosition,
                    boolean isExpanded, View convertView,
                    ViewGroup parent)
                {
                    TextView textView = getTextView();
                    textView.setText(getGroup(groupPosition)
                        .toString());
                    return textView;
                }
                @Override
                public boolean isChildSelectable(int groupPosition,
                    int childPosition)
                {
                    return true;
                }
                @Override
                public boolean hasStableIds()
                {
                    return true;
                }
            };
            // 为 ExpandableListView 设置 Adapter 对象
            list.setAdapter(adapter);
            // 使用对话框来显示查询结果
            new AlertDialog.Builder(ContactProviderTest.this)
                .setView(resultDialog).setPositiveButton("确定", null)
                .show();
        }
    });
```

上面的程序中第一行粗体字代码使用 ContentResolver 向 ContactsContract. Contacts. CONTENT_URI 查询数据，将可以把系统中所有联系人信息查询出来，程序中第二行粗体字代码使用 ContentResolver 向 ContactsContract.CommonDataKinds.Phone. CONTENT_ URI 查询数据，用于查询指定联系人的电话信息；第三行粗体字代码使用 ContentResolver 向 ContactsContract.Common DataKinds.Email. CONTENT_URI 查询数据，用于查询指定联系人

的 Email 信息。

上面的程序通过 ContentResolver 将联系人信息、电话信息、E-mail 信息查询出来之后，使用了一个 ExpandableListView 来显示所有联系人信息。

运行该程序，并单击“查询”按钮，程序将会使用 ExpandableListView 显示所有联系人信息，如图 9.4 所示。



图 9.4 查询联系人信息

需要指出的是，上面的应用程序需要读取、添加联系人信息，因此要记得在 AndroidManifest.xml 文件中为该应用程序授权。也就是在该文件的根元素中添加如下元素：

```
<!-- 授予读联系人 ContentProvider 的权限 -->
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<!-- 授予写联系人 ContentProvider 的权限 -->
<uses-permission android:name="android.permission.WRITE_CONTACTS"/>
```

该程序界面上提供了三个文本框，用户可以在这三个文本框中输入联系人名字、电话号码、E-mail 地址，然后单击“添加”按钮。该按钮绑定事件监听器的代码如下。

程序清单：codes\09\9.3\ContactProviderTest\src\org\crazyit\content\ContactProviderTest.java

```
// 为 add 按钮的单击事件绑定监听器
add.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 获取程序界面中的三个文本框
        String name = ((EditText) findViewById(R.id.name))
            .getText().toString();
        String phone = ((EditText) findViewById(R.id.phone))
            .getText().toString();
        String email = ((EditText) findViewById(R.id.email))
            .getText().toString();
        // 创建一个空的 ContentValues
        ContentValues values = new ContentValues();
        // 向 RawContacts.CONTENT_URI 执行一个空值插入
        // 目的是获取系统返回的 rawContactId
        Uri rawContactUri = getContentResolver().insert(
            RawContacts.CONTENT_URI, values);
        long rawContactId = ContentUris.parseId(rawContactUri);
        values.clear();
        values.put(Data.RAW_CONTACT_ID, rawContactId);
        // 设置内容类型
        values
            .put(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE);
        // 设置联系人名字
        values.put(StructuredName.GIVEN_NAME, name);
        // 向联系人 URI 添加联系人名字
        getContentResolver().insert(
            android.provider.ContactsContract.Data.CONTENT_URI,
            values);
        values.clear();
        values.put(Data.RAW_CONTACT_ID, rawContactId);
        values.put(Data.MIMETYPE, Phone.CONTENT_ITEM_TYPE);
        // 设置联系人的电话号码
        values.put(Phone.NUMBER, phone);
        // 设置电话类型
        values.put(Phone.TYPE, Phone.TYPE_MOBILE);
```

```
            // 向联系人电话号码 URI 添加电话号码
            getContentResolver().insert(
                android.provider.ContactsContract.Data.CONTENT_URI,
                values);
            values.clear();
            values.put(Data.RAW_CONTACT_ID, rawContactId);
            values.put(Data.MIMETYPE, Email.CONTENT_ITEM_TYPE);
            // 设置联系人的 E-mail 地址
            values.put(Email.DATA, email);
            // 设置该电子邮件的类型
            values.put(Email.TYPE, Email.TYPE_WORK);
            // 向联系人 E-mail URI 添加 E-mail 数据
            getContentResolver().insert(
                android.provider.ContactsContract.Data.CONTENT_URI,
                values);
            Toast.makeText(ContactProviderTest.this, "联系人数据添加成功",
                Toast.LENGTH_LONG).show();
        }
    });
```

上面的程序中第一行粗体字代码通过 ContentResolver 添加一个联系人记录，第二行粗体字代码通过 ContentResolver 为指定联系人添加一个电话号码；第三行粗体字代码通过 ContentResolver 为指定联系人添加一个 E-mail 地址。

如果用户在程序的三个文本框中分别输入联系人姓名、电话号码、E-mail 地址，并单击“添加”按钮，将可以看到该联系人信息被添加到系统中。启动 Android 系统的 Contacts 程序，可看到如图 9.5 所示的界面。

单击图 9.5 所示界面中“沙和尚”列表项，系统将打开该联系人的联系方式详情，如图 9.6 所示。



图 9.5　通过 ContentResolver 添加的联系人



图 9.6　通过 ContentResolver 添加的联系方式详情

从图 9.6 中可以看出，前面在程序中加的 mobile 类型的电话号码、work 类型的 E-mail 地址都显示出来了，这就表明通过 ContentResolver 添加联系人成功。



提示：

可能有读者对 mobile 类型的电话号码、work 类型的 E-mail 地址感到迷惑，这是因为一个联系人可能有多个电话号码，比如移动电话（mobile 类型）、工作电话（work 类型）、家庭电话（home 类型）等，E-mail 地址也与此类似。实际上，好像只有早期手机上每个联系人只能添加一个电话，现在的手机应该都支持为一个联系人关联多个电话号码。

9.3.2 使用 ContentProvider 管理多媒体内容

Android 提供了 Camera 程序来支持拍照、拍摄视频，用户拍摄的照片、视频都将存放在固定的位置。有些时候，其他应用程序可能需要直接访问 Camera 所拍摄的照片、视频等，为了处理这种需求，Android 同样为这些多媒体内容提供了 ContentProvider。

Android 为多媒体提供的 ContentProvider 的 Uri 如下。

- MediaStore.Audio.Media.EXTERNAL_CONTENT_URI：存储在外部存储器（SD卡）上的音频文件内容的 ContentProvider 的 Uri。
- MediaStore.Audio.Media.INTERNAL_CONTENT_URI：存储在手机内部存储器上的音频文件内容的 ContentProvider 的 Uri。
- MediaStore.Images.Media.EXTERNAL_CONTENT_URI：存储在外部存储器（SD卡）上的图片文件内容的 ContentProvider 的 Uri。
- MediaStore.Images.Media.INTERNAL_CONTENT_URI：存储在手机内部存储器上的图片文件内容的 ContentProvider 的 Uri。
- MediaStore.Video.Media.EXTERNAL_CONTENT_URI：存储在外部存储器（SD卡）上的音频文件内容的 ContentProvider 的 Uri。
- MediaStore.Video.Media.INTERNAL_CONTENT_URI：存储在手机内部存储器上的音频文件内容的 ContentProvider 的 Uri。

该示例程序界面中包含两个按钮，一个“查看”按钮，用于查看多媒体数据中的所有图片，一个“添加”按钮，用于向多媒体数据中添加图片。

该程序中“查看”按钮所绑定的事件监听器的代码如下。

程序清单：codes\09\9.3\MediaProviderTest\src\org\crazyit\content\MediaProviderTest.java

```
// 为 view 按钮的单击事件绑定监听器
view.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 清空 names、descs、fileNames 集合里原有的数据
        names.clear();
        descs.clear();
        fileNames.clear();
        // 通过 ContentResolver 查询所有图片信息
        Cursor cursor = getContentResolver().query(
            Media.EXTERNAL_CONTENT_URI, null, null, null, null);
        while (cursor.moveToNext())
        {
            // 获取图片的显示名
            String name = cursor.getString(cursor
                .getColumnIndex(Media.DISPLAY_NAME));
            // 获取图片的详细描述
            String desc = cursor.getString(cursor
                .getColumnIndex(Media.DESCRIPTION));
            // 获取图片的保存位置的数据
            byte[] data = cursor.getBlob(cursor
                .getColumnIndex(Media.DATA));
            // 将图片名添加到 names 集合中
            names.add(name);
            // 将图片描述添加到 descs 集合中
            descs.add(desc);
```

```
            // 将图片保存路径添加到 fileNames 集合中
            fileNames.add(new String(data, 0, data.length - 1));
        }
        // 创建一个 List 集合，List 集合的元素是 Map
        List<Map<String, Object>> listItems =
            new ArrayList<Map<String, Object>>();
        // 将 names、descs 两个集合对象的数据转换到 Map 集合中
        for (int i = 0; i < names.size(); i++)
        {
            Map<String, Object> listItem =
                new HashMap<String, Object>();
            listItem.put("name", names.get(i));
            listItem.put("desc", descs.get(i));
            listItems.add(listItem);
        }
        // 创建一个 SimpleAdapter
        SimpleAdapter simpleAdapter = new SimpleAdapter(
            MediaProviderTest.this, listItems
            , R.layout.line,
            new String[] { "name", "desc" }
            , new int[] {R.id.name, R.id.desc });
        // 为 show ListView 组件设置 Adapter
        show.setAdapter(simpleAdapter);
    }
});
```



图 9.7　查看系统图片列表

该程序中粗体字代码使用 ContentResolver 向 MediaStore.Audio.Images.EXTERNAL_CONTENT_URI 查询数据，这将查询出所有位于外部存储器上的图片信息。查询出图片信息之后，该程序使用了一个 ListView 来显示这些图片信息。

当用户单击“查看”按钮之后，系统将会显示如图 9.7 所示列表。

注意图 9.7 所示列表中的列表项，该程序为该列表的列表项被单击绑定了如下事件监听器。

程序清单：codes\09\9.3\MediaProviderTest\src\org\crazyit\content\MediaProviderTest.java

```
// 为 show ListView 的列表项单击事件添加监听器
show.setOnItemClickListener(new OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> parent
        , View source, int position, long id)
    {
        // 加载 view.xml 界面布局代表的视图
        View viewDialog = getLayoutInflater().inflate(
            R.layout.view, null);
        // 获取 viewDialog 中 ID 为 image 的组件
        ImageView image = (ImageView) viewDialog
            .findViewById(R.id.image);
        // 设置 image 显示指定图片
        image.setImageBitmap(BitmapFactory.decodeFile(
            fileNames.get(position)));
        // 使用对话框显示用户单击的图片
        new AlertDialog.Builder(MediaProviderTest.this)
            .setView(viewDialog).setPositiveButton("确定", null)
            .show();
    }
});
```

正如上面的程序中粗体字代码所示，当用户单击指定列表项之后，程序会使用一个包含 ImageView 的对话框来显示该类表项所对应的图片。当用户单击指定列表项之后，系统将显示图 9.8 所示对话框。



图 9.8 查看图片



提示：

通过该实例程序，可以非常方便地实现 Android 图片浏览器：Android 图片浏览器并不需要程序员去遍历 SD 卡的图片文件，直接查询系统多媒体 ContentProvider 即可获取系统中所有的图片信息。

该程序中“添加”按钮用于将本程序中指定图片添加到多媒体数据中——实际上完全可以把任意图片添加到多媒体数据中。程序为“添加”按钮绑定事件监听器的代码如下。

程序清单：codes\09\9.3\MediaProviderTest\src\org\crazyit\content\MediaProviderTest.java

```
// 为 add 按钮的单击事件绑定监听器
add.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 创建 ContentValues 对象，准备插入数据
        ContentValues values = new ContentValues();
        values.put(Media.DISPLAY_NAME, "jinta");
        values.put(Media.DESCRIPTION, "金塔");
        values.put(Media.MIME_TYPE, "image/jpeg");
        // 插入数据，返回所插入数据对应的 Uri
        Uri uri = getContentResolver().insert(
            Media.EXTERNAL_CONTENT_URI, values);
        // 加载应用程序下的 jinta 图片
        Bitmap bitmap = BitmapFactory.decodeResource(
            MediaProviderTest.this.getResources(),
            R.drawable.jinta);
        OutputStream os = null;
        try
        {
            // 获取刚插入的数据的 Uri 对应的输出流
            os = getContentResolver().openOutputStream(uri); //①
            // 将 bitmap 图片保存到 Uri 对应的数据节点中
            bitmap.compress(Bitmap.CompressFormat.JPEG, 100, os);
            os.close();
        }
        catch (IOException e)
```

```
                {
                    e.printStackTrace();
                }
            }
        });
```

上面的程序中粗体字代码使用了 ContentResolver 向 MediaStore.Audio.Images.EXTERNAL_CONTENT_URI 插入一条记录，但此时还未把真正的图片数据插入进去。程序①号粗体字代码打开了刚插入图片的 Uri 对应的 OutputStream，接下来程序调用 Bitmap 的 compress 方法把实际的图片内容保存到 OutputStream 中，这样就会把实际图片内容存入系统。

9.4 监听 ContentProvider 的数据改变

前面介绍的是当 ContentProvider 将数据共享出来之后，ContentResolver 会根据业务需要去主动查询 ContentProvider 所共享数据；在有些时候，应用程序需要实时监听 ContentProvider 所共享数据的改变，并随着 ContentProvider 的数据的改变而提供响应，这就需要利用 ContentObserver 了。

9.4.1 ContentObserver 简介

前面介绍开发 ContentProvider 时，不管实现 insert、delete、update 方法中的哪一个，只要该方法导致了 ContentProvider 里数据的改变，程序就调用了如下代码：

```
getContext().getContentResolver().notifyChange(uri, null);
```

这行代码可用于通知所有注册在该 Uri 上的监听者：该 ContentProvider 所共享的数据发生了改变。

为了在应用程序中监听 ContentProvider 数据的改变，需要利用 Android 提供 ContentObserver 基类。

监听 ContentProvider 数据改变的监听器需要继承 ContentObserver 类，并重写该基类所定义的 onChange(boolean selfChange)方法——当它所监听的 ContentProvider 的数据发生改变时，该 onChange 将会被触发。

为了监听指定 ContentProvider 的数据变化，需要通过 ContentResolver 向指定 Uri 注册 ContentObserver 监听器。ContentResolver 提供了如下方法来注册监听器：

registerContentObserver (Uri uri, boolean notifyForDescendents, ContentObserver observer)。该方法中三个参数的说明如下。

- uri：该监听器所监听的 ContentProvider 的 Uri。
- notifyForDescendents：如果该参数设为 true，假如注册监听的 Uri 为 content://abc，那么 Uri 为 content://abc/xyz、content://abc/xyz/foo 的数据改变时也会触发该监听器；如果该参数设为 false，假如注册监听的 Uri 为 content://abc，那么只有 content://abc 的数据发生改变时会触发该监听器。
- observer：监听器实例。

例如如下代码片段可用于为指定 Uri 注册监听器：

```
getContentResolver().registerContentObserver(Uri.parse("content://sms")
    , true, new SmsObserver(new Handler()));
```

上面的代码中 SmsObserver 就是 ContentObserver 的子类。

实例：监听用户发出的短信

本实例通过监听 Uri 为 content://sms 的数据改变即可监听到用户短信的数据改变，并在监听器的 onChange(boolean selfChange)方法里查询 Uri 为 content://sms/outbox 的数据，这样即可获取用户正在发送的短信（用户正在发送的短信保存在发件箱内）。

该程序代码如下。

程序清单：codes\09\9.4\MonitorSms\src\org\crazyit\content\MonitorSms.java

```
public class MonitorSms extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 为 content://sms 的数据改变注册监听器
        getContentResolver().registerContentObserver(
            Uri.parse("content://sms"), true,
            new SmsObserver(new Handler()));
    }
    // 提供自定义的 ContentObserver 监听器类
    private final class SmsObserver extends ContentObserver
    {
        public SmsObserver(Handler handler)
        {
            super(handler);
        }
        public void onChange(boolean selfChange)
        {
            // 查询发送箱中的短信（处于正在发送状态的短信放在发送箱）
            Cursor cursor = getContentResolver().query(
                Uri.parse("content://sms/outbox")
                , null, null, null, null);
            // 遍历查询得到的结果集，即可获取用户正在发送的短信
            while (cursor.moveToNext())
            {
                StringBuilder sb = new StringBuilder();
                // 获取短信的发送地址
                sb.append("address=").append(cursor
                        .getString(cursor.getColumnIndex("address")));
                // 获取短信的标题
                sb.append(";subject=").append(cursor
                        .getString(cursor.getColumnIndex("subject")));
                // 获取短信的内容
                sb.append(";body=").append(cursor
                        .getString(cursor.getColumnIndex("body")));
                // 获取短信的发送时间
                sb.append(";time=").append(cursor
                        .getLong(cursor.getColumnIndex("date")));
                System.out.println("Has Sent SMS::" + sb.toString());
            }
        }
    }
}
```

上面的程序中第一行粗体字代码用于监听 Uri 为 content://sms 处的数据改变，就可以监

听到用户短信数据的改变了；上面的程序中第二行粗体字代码用于查询 content://sms/outbox 的全部数据，也就是查询发件箱（正在发送的短信保存在发件箱内）内全部短信，这样就可获取用户正在发送的短信的详情。

运行该程序，在不关闭该程序的情况下打开 Android 系统内置的“Messaging”程序来发送短信——直接向本机号码发送即可。当用户发送短信时，可以在 DDMS 的 LogCat 面板看到如图 9.9 所示的输出。



图 9.9　监听用户发送的短信

这个监听用户发送短信的程序采用 Activity 来实现并不合适——因为用户必须先主动打开该 Activity，并保持该 Activity 不关闭的情况下，用户所发送的短信才会被监听到。这显然不太符合实际需求场景，在实际情况下，可能更希望该程序以后台进程的方式“不知不觉”地运行，这就需要利用 Android 的 Service 组件，接下来将介绍 Android 提供的 Service 组件。

9.5　本章小结

本章主要介绍了 Android 系统中 ContentProvider 组件的功能和用法，ContentProvider 的本质就是像一个“小网站”，它可以把应用程序的数据按照“固定规范”暴露出来，其他应用程序就可通过 ContentProvider 暴露的接口来操作内部的数据了。可以这样说，ContentProvider 是 Android 系统内不同程序之间进行数据交换的标准接口。学习本章需要重点掌握三个 API 的用法：ContentResolver、ContentProvider 和 ContentObserver，其中 ContentResolver 用于操作 ContentProvider 提供的数据，ContentObserver 用于监听 ContentProvider 的数据改变；而 ContentProvider 则是所有 ContentProvider 组件的基类。

CHAPTER

10

第10章 Service与BroadcastReceiver

本章要点

- Service 组件的作用和意义
- 创建、配置 Service
- 启动、停止 Service
- 绑定本地 Service 并与之通信
- Service 的生命周期
- IntentService 的功能和用法
- 开发远程 AIDL Service
- 在客户端程序中调用远程 AIDL Service
- TelephonyManager 的功能和用法
- 监听手机电话
- SmsManager 的功能和用法
- 监听手机短信
- AudioManager 的功能和用法
- Vibrator 的功能和用法
- AlarmManager 的功能和用法
- BroadcastReceiver 组件的作用和意义
- 开发、配置 BroadcastReceiver 组件
- 发送广播、发送有序广播
- 使用 BroadcastReceiver 接收系统广播

Service 是 Android 四大组件中与 Activity 最相似的组件，它们都代表可执行的程序，Service 与 Activity 的区别在于：Service 一直在后台运行，它没有用户界面，所以绝不会到前台来。一旦 Service 被启动起来之后，它就与 Activity 一样。它完全具有自己的生命周期。关于程序中 Activity 与 Service 的选择标准是：如果某个程序组件需要在运行时向用户呈现某种界面，或者该程序需要与用户交互，就需要使用 Activity，否则就应该考虑使用 Service 了。

开发者开发 Service 的步骤与开发 Activity 的步骤很像，开发 Service 组件需要先开发一个 Service 的子类，然后在 AndroidManifest.xml 文件中配置该 Service，配置时可通过<intent-filter.../>元素指定它可被哪些 Intent 启动。

Android 系统本身提供了大量的 Service 组件，开发者可通过这些系统 Service 来操作 Android 系统本身。

本章还将向读者介绍 BroadcastReceiver 组件。BroadcastReceiver 组件就像一个全局的事件监听器，只不过它用于监听系统发出的 Broadcast。通过使用 BroadcastReceiver，即可在不同应用程序之间通信。

10.1 Service 简介

Service 组件也是可执行的程序，它也有自己的生命周期。创建、配置 Service 与创建、配置 Activity 的过程基本相似，下面详细介绍 Android Service 的开发。

➤➤10.1.1 创建、配置 Service

就像开发 Activity 需要两个步骤：① 开发 Activity 子类；② 在 AndroidManifest.xml 文件中配置 Activity，开发 Service 也需要两个步骤：

1. 定义一个继承 Service 的子类。
2. 在 AndroidManifest.xml 文件中配置该 Service。

Service 与 Activity 还有一点相似之处，它们都是从 Context 派生出来的，因此它们都可调用 Context 里定义的如 getResources()、getContentResolver()等方法。

与 Activity 相似的是，Service 中也定义了系列生命周期方法，如下所示。

- IBinder onBind(Intent intent)：该方法是 Service 子类必须实现的方法。该方法返回一个 IBinder 对象，应用程序可通过该对象与 Service 组件通信。
- void onCreate()：当该 Service 第一次被创建后将立即回调该方法。
- void onDestroy()：当该 Service 被关闭之前将会回调该方法。
- void onStartCommand(Intent intent, int flags, int startId)：该方法的早期版本是 void onStart (Intent intent, int startId)，每次客户端调用 startService(Intent)方法启动该 Service 时都会回调该方法。
- boolean onUnbind(Intent intent)：当该 Service 上绑定的所有客户端都断开连接时将会回调该方法。

下面的类定义了第一个 Service 组件。

程序清单：codes\10\10.1\FirstService\src\org\crazyit\service\FirstService.java

```
public class FirstService extends Service
{
```

```
        // 必须实现的方法
        @Override
        public IBinder onBind(Intent arg0)
        {
            return null;
        }
        // Service 被创建时回调该方法
        @Override
        public void onCreate()
        {
            super.onCreate();
            System.out.println("Service is Created");
        }
        // Service 被启动时回调该方法
        @Override
        public int onStartCommand(Intent intent, int flags, int startId)
        {
            System.out.println("Service is Started");
            return START_STICKY;
        }
        // Service 被关闭之前回调
        @Override
        public void onDestroy()
        {
            super.onDestroy();
            System.out.println("Service is Destroyed");
        }
}
```

上面这个 Service 什么也没干——它只是重写了 Service 组件的 onCreate()、onStartCommand()、onDestroy()、onBind()等方法，重写这些方法时只是简单地输出了一条字符串。



提示： 虽然这个 Service 什么都没干，但实际上它是 Service 组件的框架，如果希望 Service 组件做某些事情，那么只要在 onCreate()或 onStartCommand()方法中定义相关业务代码即可。

定义了上面的 Service 之后，接下来需要在 AndroidManifest.xml 文件中配置该 Service，配置 Service 使用<service.../>元素。与配置 Activity 相似的是，配置 Service 时也可为<service.../>元素配置<intent-filter.../>子元素，用于说明该 Service 可被哪些 Intent 启动。

在 AndroidManifest.xml 文件中增加如下配置片段来配置该 Service：

```
<!-- 配置一个 Service 组件 -->
<service android:name=".FirstService">
    <intent-filter>
        <!-- 为该 Service 组件的 intent-filter 配置 action -->
        <action android:name="org.crazyit.service.FIRST_SERVICE" />
    </intent-filter>
</service>
```

从上面的配置片段不难看出，配置 Service 与配置 Activity 的差别并不大，只是配置 Service 使用<service.../>元素，而且无须指定 android:label 属性——因为 Service 没有界面，总是位于后台运行，为该 Service 指定标签没有太大的意义。

当该 Service 开发完成之后，接下来就可在程序中运行该 Service 了，Android 系统中运

行 Service 有如下两种方式。

- 通过 Context 的 startService()方法：通过该方法启用 Service，访问者与 Service 之间没有关联，即使访问者退出了，Service 仍然运行。
- 通过 Context 的 bindService()方法：使用该方法启用 Service，访问者与 Service 绑定在了一起，访问者一旦退出，Service 也就终止。

下面先示范的第一种方式运行 Service。

10.1.2 启动和停止 Service

下面的程序使用 Activity 作为 Service 的访问者，该 Activity 的界面中包含两个按钮，一个按钮用于启动 Serivce，一个按钮用于关闭 Service。

该 Activity 的代码如下。

程序清单：codes\10\10.1\FirstService\src\org\crazyit\service\StartServiceTest.java

```
public class StartServiceTest extends Activity
{
    Button start, stop;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的 start、stop 两个按钮
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        // 创建启动 Service 的 Intent
        final Intent intent = new Intent();
        // 为 Intent 设置 Action 属性
        intent.setAction("org.crazyit.service.FIRST_SERVICE");
        start.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 启动指定 Serivce
                startService(intent);
            }
        });
        stop.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 停止指定 Serivce
                stopService(intent);
            }
        });
    }
}
```

从上面程序的粗体字代码不难看出，启动、关闭 Service 十分简单，调用 Context 里定义的 startService()、stopSerivce()方法即可启动、关闭 Service。

运行该程序，通过程序界面先启动 Service、再关闭 Service 将可以在 DDMS 的 LogCat 面板看到如图 10.1 所示的输出。



图 10.1 启动、关闭 Service

如果在不关闭 Service 的情况下，连续三次单击“启动 Service”按钮，程序将会连续三次启动 Service，此时在 DDMS 的 LogCat 面板可看到如图 10.2 所示的输出。



图 10.2 连续启动 Service

从图 10.2 可以看出，每当 Service 被创建时会回调 onCreate 方法，每次 Service 被启动时都会回调 onStart 方法——多次启动一个已有的 Service 组件将不会再回调 onCreate 方法，但每次启动时都会回调 onStartCommand()方法。

10.1.3 绑定本地 Service 并与之通信

当程序通过 startService()和 stopService()启动、关闭 Service 时，Service 与访问者之间基本上不存在太多的关联，因此 Service 和访问者之间也无法进行通信、数据交换。

如果 Service 和访问者之间需要进行方法调用或数据交换，则应该使用 bindService()和 unbindService()方法启动、关闭 Service。

Context 的 bindService()方法的完整方法签名为：bindService(Intent service, ServiceConnection conn, int flags)，该方法的三个参数的解释如下。

- service：该参数通过 Intent 指定要启动的 Service。
- conn：该参数是一个 ServiceConnection 对象，该对象用于监听访问者与 Service 之间的连接情况。当访问者与 Service 之间连接成功时将回调该 ServiceConnection 对象的 onServiceConnected(ComponentName name, IBinder service)方法；当 Service 所在的宿主进程由于异常中止或由于其他原因终止，导致该 Service 与访问者之间断开连接时回调该 ServiceConnection 对象的 onServiceDisconnected (ComponentName name)方法。

注意：

当调用者主动通过 unBindService()方法断开与 Service 的连接时，ServiceConnection 对象的 onServiceDisconnected (ComponentName name)方法并不会被调用。



- flags：指定绑定时是否自动创建 Service（如果 Service 还未创建）。该参数可指定为 0（不自动创建）或 BIND_AUTO_CREATE（自动创建）。

注意到 ServiceConnection 对象的 onServiceConnected 方法中有一个 IBinder 对象，该对象即可实现与被绑定 Service 之间的通信。

当开发 Service 类时，该 Service 类必须提供一个 IBinder onBind(Intent intent)方法，在绑定本地 Service 的情况下，onBind(Intent intent)方法所返回的 IBinder 对象将会传给 ServiceConnection 对象里 onServiceConnected(ComponentName name, IBinder service)方法的 service 参数，这样访问者就可通过该 IBinder 对象与 Service 进行通信。



提示：

IBinder 对象相当于 Service 组件的内部钩子，该钩子关联到绑定的 Service 组件，当其他程序组件绑定该 Service 时，Service 将会把 IBinder 对象返回给其他程序组件，其他程序组件通过该 IBinder 对象即可与 Service 组件进行实时通信。

实际上开发时通常会采用继承 Binder（IBinder 的实现类）的方式实现自己的 IBinder 对象。

下面的程序示范了如何在 Acticity 中绑定本地 Service，并获取 Service 的运行状态。该程序的 Service 类需要“真正”实现 onBind()方法，并让该方法返回一个有效的 IBinder 对象，该 Service 类的代码如下。

程序清单：codes\10\10.1\BindService\src\org\crazyit\service\BindService.java

```
public class BindService extends Service
{
    private int count;
    private boolean quit;
    // 定义 onBinder 方法所返回的对象
    private MyBinder binder = new MyBinder();
    // 通过继承 Binder 来实现 IBinder 类
    public class MyBinder extends Binder //①
    {
        public int getCount()
        {
            // 获取 Service 的运行状态：count
            return count;
        }
    }
    // 必须实现的方法，绑定该 Service 时回调该方法
    @Override
    public IBinder onBind(Intent intent)
    {
        System.out.println("Service is Binded");
        // 返回 IBinder 对象
        return binder;
    }
    // Service 被创建时回调该方法
    @Override
    public void onCreate()
    {
        super.onCreate();
        System.out.println("Service is Created");
        // 启动一条线程，动态地修改 count 状态值
        new Thread()
        {
            @Override
            public void run()
            {
```

```
                while (!quit)
                {
                    try
                    {
                        Thread.sleep(1000);
                    }
                    catch (InterruptedException e)
                    {
                    }
                    count++;
                }
            }
        }.start();
    }
    // Service被断开连接时回调该方法
    @Override
    public boolean onUnbind(Intent intent)
    {
        System.out.println("Service is Unbinded");
        return true;
    }
    // Service被关闭之前回调该方法
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        this.quit = true;
        System.out.println("Service is Destroyed");
    }
}
```

上面Service类的粗体字代码实现了onBind()方法，该方法返回了一个可访问该Service状态数据（count值）的IBinder对象，该对象将被传给该Service的访问者。

上面程序的①号代码通过继承Binder类实现了一个IBinder对象，这个MyBinder类是Service的内部类，这对于绑定本地Service并与之通信的场景是一种常见的情形。

接下来定义一个Activity来绑定该Service，并在该Activity中通过MyBinder对象访问Service的内部状态。该Activity的界面上包含三个按钮，第一个按钮用于绑定Service，第二个按钮用于解除绑定；第三个按钮则用于获取Service的运行状态。该Activity的代码如下。

程序清单：codes\10\10.1\BindService\src\org\crazyit\service\BindServiceTest.java

```
ppublic class BindServiceTest extends Activity
{
    Button bind, unbind, getServiceStatus;
    // 保持所启动的Service的IBinder对象
    BindService.MyBinder binder;
    // 定义一个ServiceConnection对象
    private ServiceConnection conn = new ServiceConnection()
    {
        // 当该Activity与Service连接成功时回调该方法
        @Override
        public void onServiceConnected(ComponentName name
            , IBinder service)
        {
            System.out.println("--Service Connected--");
            // 获取Service的onBind方法所返回的MyBinder对象
            binder = (BindService.MyBinder) service; //①
        }
        // 当该Activity与Service断开连接时回调该方法
```

```
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
            System.out.println("--Service Disconnected--");
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的 start、stop、getServiceStatus 按钮
        bind = (Button) findViewById(R.id.bind);
        unbind = (Button) findViewById(R.id.unbind);
        getServiceStatus = (Button) findViewById(R.id.getServiceStatus);
        // 创建启动 Service 的 Intent
        final Intent intent = new Intent();
        // 为 Intent 设置 Action 属性
        intent.setAction("org.crazyit.service.BIND_SERVICE");
        bind.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 绑定指定 Serivce
                bindService(intent, conn, Service.BIND_AUTO_CREATE);
            }
        });
        unbind.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 解除绑定 Serivce
                unbindService(conn);
            }
        });
        getServiceStatus.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取、并显示 Service 的 count 值
                Toast.makeText(BindServiceTest.this,
                    "Serivce 的 count 值为：" + binder.getCount(),
                    Toast.LENGTH_SHORT).show(); //②
            }
        });
    }
}
```

上面的程序中①号代码用于在该 Activity 与 Service 连接成功时获取 Service 的 onBind() 方法所返回的 MyBinder 对象；程序的②号代码即可通过 MyBinder 对象来访问 Service 的运行状态了。

运行该程序，单击程序界面中“绑定 Service”按钮，即可看到 DDMS 的 LogCat 有如图 10.3 所示的输出。

在该 Activity 中绑定该 Service 之后，该 Activity 还可通过 MyBinder 对象来获取 Service 的运行状态，如果用户单击程序界面上的“获取 Service 状态”按钮即可看到如图 10.4 所示

的输出。



图 10.3 绑定 Service



图 10.4 访问 Service 的运行状态

从图 10.4 所示的输出可以看到，该 Activity 可以非常方便地访问到 Service 的运行状态。虽然本程序只是一个简单的示例，该 Activity 只是访问了 Service 的一个简单 count 值，但实际上完全可以让 MyBinder 去操作 Service 中更多的数据——到底需要访问 Service 的多少数据，完全取决于实际业务的需要。



提示： 对于 Service 的 onBind()方法所返回的 IBinder 对象来说，它可被当成该 Service 组件所返回的代理对象，Service 允许客户端通过该 IBinder 对象来访问 Service 内部的数据，这样即可实现客户端与 Service 之间的通信。

如果我们单击程序界面上的“解除绑定”按钮，即可在 DDMS 的 LogCat 中看到如图 10.5 所示的输出。



图 10.5 解除绑定

正如图 10.5 中所示，当程序调用 unbindService()方法解除对某个 Service 的绑定时，系统会先回调该 Service 的 onUnbind()方法，然后再回调 onDestroy()方法。

与多次调用 startService()方法启动 Service 不同的是，多次调用 bindService()方法并不会执行重复绑定。对于前一个示例程序，用户每单击“启动 Serivce”按钮一次，系统就会回调 Service 的 onStartCommand()方法一次；对于这个示例程序，不管用户单击“绑定 Service”多少次，系统只会回调 Service 的 onBind()方法一次。

10.1.4 Service 的生命周期

通过前面两个示例，读者应该已经大致明白 Service 的生命周期了。随着应用程序启动 Service 方式的不同，Service 的生命周期也略有差异。

如果应用程序通过 startService()方法来启动 Service，Service 的生命周期如图 10.6 左边所示。

如果应用程序通过 bindService()方法来启动 Service，Service 的生命周期如图 10.6 右边所示。

Service 生命周期还有一种特殊的情形，如果 Service 已由某个客户端通过 startService()

方法启动了，接下来其他客户端再调用 bindService()方法来绑定该 Service 后，再调用 unbindService()方法解除绑定，最后又调用了 bindService()方法再次绑定到 Service，这个过程所触发的生命周期方法如下。

onCreate()→onStartCommand()→onBind()→onUnbind()[重写该方法时返回了 true]→onRebind()

在上面这个触发过程中，onCreate()是创建该 Service 后立即调用的，只有当该 Service 被创建时才会被调用；onStartCommand ()方法则由客户端调用 startService()方法时触发的。图 10.7 的 LogCat 显示了上面生命周期的输出。

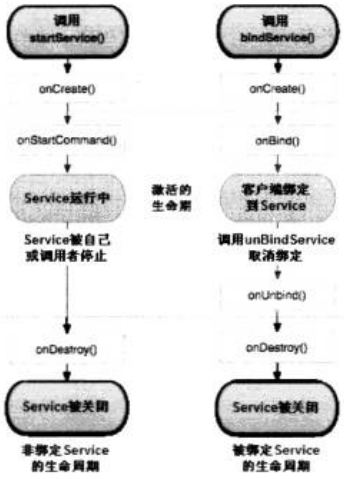


图 10.6　Service 的生命周期



图 10.7　先启动、再绑定的 Service 的生命周期

注意：

在图 10.7 所示输出中，可以看到 Service 的 onRebind()方法被回调了。如果希望该方法被回调，除了需要该 Service 是由 Activity 的 startService()方法启动之外，还需要 Service 子类重写 onUnbind()方法时返回 true。



在图 10.7 所示的输出效果中，并没有发现 Service 回调 onDestroy()方法，这是因为该 Service 并不是由 Activity 通过 bindService()方法来启动的（该 Service 事先已由 Activity 通过 startService()方法启动了），因此当 Activity 调用 unBindService()方法取消与该 Service 的绑定时，该 Service 也不会终止。

由此可见，当 Activity 调用 bindService()绑定一个已启动的 Service 时，系统只是把 Service 内部 IBinder 对象传给 Activity，并不会把该 Service 生命周期完全“绑定”到该 Activity，因而当 Activity 调用 unBindService()方法取消与该 Service 的绑定时，也只是切断该 Activity 与 Service 之间的关联，并不能停止该 Service 组件。

➤➤10.1.5　使用 IntentService

IntentService 是 Service 的子类，因此它不是普通的 Service，它比普通 Service 增加了额外的功能。

先看 Service 本身存在的两个问题：

- Service 不会专门启动一条单独的进程，Service 与它所在应用位于同一个进程中。
- Service 也不是专门一条新的线程，因此不应该在 Service 中直接处理耗时的任务。



提示：

如果开发者需要在 Service 处理耗时任务，建议在 Service 中另外启动一条新线程来处理该耗时任务。就像前面 BindService 中看到的，程序在 BindService 中的 onCreate()方法中启动了一条新线程来处理耗时任务。可能有读者感到疑惑：直接在其他程序组件中启动子线程来处理耗时任务不行吗？这种方式也不可靠，由于 Activity 可能会被用户退出，BroadcastReceiver 的生命周期本身就很短。可能出现的情况是：在子线程还没有结束的情况下，Activity 已经被用户退出了，或者 BroadcastReceiver 已经结束了。在 Activity 已经退出、BroadcastReceiver 已经结束的情况下，此时它们所在的进程就变成了空进程（没有任何活动组件的进程），系统需要内存时可能会优先终止该进程。如果宿主进程被终止，那么该进程内的所有子线程也会被中止，这样就可能导致子线程无法执行完成。

而 IntentService 正好可以弥补 Service 的上述两个不足：IntentService 将会使用队列来管理请求 Intent，每当客户端代码通过 Intent 请求启动 IntentService 时，IntentService 会将该 Intent 加入队列中，然后开启一条新的 worker 线程来处理该 Intent。对于异步的 startService()请求，IntentService 会按次序依次处理队列中的 Intent，该线程保证同一时刻只处理一个 Intent。由于 IntentService 使用新的 worker 线程处理 Intent 请求，因此 IntentService 不会阻塞主线程，所以 IntentService 自己就可以处理耗时任务。

归纳起来，IntentService 具有如下特征：

- IntentService 会创建单独的 worker 线程来处理所有的 Intent 请求。
- IntentService 会创建单独的 worker 线程来处理 onHandleIntent()方法实现的代码，因此开发者无须处理多线程问题。
- 当所有请求处理完成后，IntentService 会自动停止，因此开发者无须调用 stopSelf()方法来停止该 Service。
- 为 Service 的 onBind()方法提供了默认实现，默认实现的 onBind()方法返回 null。
- 为 Service 的 onStartCommand()方法提供了默认实现，该实现会将请求 Intent 添加到队列中。

从上面的介绍可以看出，扩展 IntentService 实现 Service 无须重写 onBind()、onStartCommand()方法，只要重写 onHandleIntent()方法即可。

下面的示例中的界面中包含了两个按钮，两个按钮分别启动普通 Service 和 IntentService，两个 Service 都需要处理耗时任务。该程序的界面布局代码很简单，主程序 Activity 代码如下。

程序清单：codes\10\10.1\IntentServiceTest\src\org\crazyit\service\IntentServiceTest.java

```
public class IntentServiceTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
    }
    public void startService(View source)
    {
        // 创建需要启动的 Service 的 Intent
        Intent intent = new Intent(this, MyService.class);
        // 启动 Service
        startService(intent);
    }
    public void startIntentService(View source)
    {
        // 创建需要启动的 IntentService 的 Intent
        Intent intent = new Intent(this, MyIntentService.class);
        // 启动 IntentService
        startService(intent);
    }
}
```

上面 Activity 的两个事件处理方法中分别启动 MyService 和 MyIntentService，其中 MyService 是继承 Service 的子类，而 MyIntentService 则是继承 IntentService 的子类。

下面是 MyService 类的代码。

程序清单：codes\10\10.1\IntentServiceTest\src\org\crazyit\service\MyService.java

```
public class MyService extends Service
{
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        // 该方法内执行耗时任务可能导致 ANR (Application Not Responding) 异常
        long endTime = System.currentTimeMillis() + 20 * 1000;
        System.out.println("onStart");
        while (System.currentTimeMillis() < endTime)
        {
            synchronized (this)
            {
                try
                {
                    wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e)
                {
                }
            }
        }
        System.out.println("---耗时任务执行完成---");
        return START_STICKY;
    }
}
```

上面 MyService 在 onStartCommand()方法中使用线程暂停的方式来模拟了耗时任务，该线程暂停了 20 秒，相当于该耗时任务需要执行 20 秒。由于普通 Service 的执行会阻塞主线程，因此启动该线程将会导致程序出现 ANR（Application Not Responding）异常。

下面是 MyIntentService 类的代码。

程序清单：codes\10\10.1\IntentServiceTest\src\org\crazyit\service\MyIntentService.java

```
public class MyIntentService extends IntentService
{
    public MyIntentService()
    {
        super("MyIntentService");
    }
    // IntentService 会使用单独的线程来执行该方法的代码
    @Override
    protected void onHandleIntent(Intent intent)
    {
        // 该方法内可以执行任何耗时任务，比如下载文件等，此处只是让线程暂停 20 秒
        long endTime = System.currentTimeMillis() + 20 * 1000;
        System.out.println("onStart");
        while (System.currentTimeMillis() < endTime)
        {
            synchronized (this)
            {
                try
                {
                    wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e)
                {
                }
            }
        }
        System.out.println("---耗时任务执行完成---");
    }
}
```

从上面的代码可以看出，MyIntentService 继承了 IntentService，并不需要实现 onBind()、onStartCommand()方法，只要实现 onHandleIntent()方法即可，在该方法中即可定义该 Service 需要完成的任务。本示例的 onHandleIntent()方法也用线程暂停的方式来模拟了耗时任务，线程同样暂停了 20 秒。但由于 IntentService 会使用单独的线程来完成该耗时任务，因此启动 MyIntentService 不会阻塞前台线程。

运行该实例，如果单击界面上的“启动普通 Service”按钮，将会激发 startService()方法，该方法将会启动 MyService 去执行耗时任务，此时将会导致程序 UI 线程被阻塞（程序界面失去响应），而且由于阻塞时间太长，因此程序将会看到如图 10.8 所示的 ANR 异常。



图 10.8 使用普通 Service 执行耗时任务导致 ANR 异常

相反，如果调用“启动 IntentService”来启动 MyIntentService，虽然 MyIntentService 也需要执行耗时任务，但由于 MyIntentService 会使用单独的 worker 线程，因此 MyIntentService 不会阻塞前台的 UI 线程，因此程序界面不会失去响应。

10.2 跨进程调用 Service（AIDL Service）

Android 系统中，各应用程序都运行在自己的进程中，进程之间一般无法直接进行数据交换。为了实现这种跨进程通信（interprocess communication，简称 IPC），Android 提供了 AIDL Service。

AIDL Service 与传统技术中 Corba、Java 技术中 RMI（远程方法调用）存在一定的相似之处。

10.2.1 AIDL Service 简介

Android 的远程 Service 调用与 Java 的 RMI 基本相似，一样都是先定义一个远程调用接口，然后为该接口提供一个实现类即可。



提示：

如果读者需要了解 Java RMI 相关的详细知识，可以参考疯狂 Java 体系的《经典 Java EE 企业应用实战》。

与 RMI 不同的是，客户端访问 Service 时，Android 并不是直接返回 Service 对象给客户端——这一点绑定本地 Service 时已经看到，Service 只是将 Service 的代理对象（IBinder 对象）通过 onBind()方法返回给客户端。因此 Android 的 AIDL 远程接口的实现类就是那个 IBinder 实现类。

与绑定本地 Service 不同的是，本地 Service 的 onBind()方法会直接把 IBinder 对象本身传给客户端的 ServiceConnection 的 onServiceConnected 方法的第二个参数。但远程 Service 的 onBind()方法只是将 IBinder 对象的代理传给客户端的 ServiceConnection 的 onServiceConnected 方法的第二个参数。

当客户端获取了远程 Service 的 IBinder 对象的代理之后，接下来就可通过该 IBinder 对象去回调远程 Service 的属性或方法了。

10.2.2 创建 AIDL 文件

与 RMI 不同的是，RMI 直接使用 Java 语言来定义远程接口，但 Android 需要 AIDL（Android Interface Definition Language）来定义远程接口。

不过读者不用担心，AIDL 接口定义语言的语法十分简单，这种接口定义语言并不是一种真正的编程语言，它只是定义两个进程之间的通信接口，因此语法非常简单。AIDL 的语法与 Java 接口很相似，但存在如下几点差异：

- AIDL 定义接口的源代码必须以.aidl 结尾。
- AIDL 接口中用到数据类型，除了基本类型、String、List、Map、CharSequence 之外，其他类型全部都需要导包，即使它们在同一个包中也需要导包。

开发人员定义的 AIDL 接口只是定义了进程之间的通信接口，Service 端、客户端都需要使用 Android SDK 安装目录下的 platform-tools 子目录下的 aidl.exe 工具为该接口提供实现。如果开发者使用 ADT 工具进行开发，那么 ADT 工具会自动为该 AIDL 接口生成实现。

例如我们在应用中定义如下 AIDL 接口代码。

程序清单：codes\10\10.2\AidlService\src\org\crazyit\service\ICat.aidl

```
package org.crazyit.service;
interface ICat
{
    String getColor();
    double getWeight();
}
```

上面的AIDL接口与Java接口的语法非常相似，因此读者无须把它想得过于复杂。

定义好上面的AIDL接口之后，ADT工具会自动在gen/org/crazyit/service目录下生成一个ICat.java接口，在该接口里包含一个Stub内部类，该内部类实现了IBinder、ICat两个接口，这个Stub类将会作为远程Service的回调类——它实现了IBinder接口，因此可作为Service的onBind()方法的返回值。

10.2.3 将接口暴露给客户端

上一步定义好AIDL接口之后，接下来就可定义一个Service实现类了，该Service的onBind()方法所返回的IBinder对象应该是ADT所生成的ICat.Stub的子类的实例。至于其他部分，则与开发本地Service完全一样。

下面是本示例的Service类代码。

程序清单：codes\10\10.2\AidlService\src\org\crazyit\service\AidlService.java

```
public class AidlService extends Service
{
    private CatBinder catBinder;
    Timer timer = new Timer();
    String[] colors = new String[]{
        "红色",
        "黄色",
        "黑色"
    };
    double[] weights = new double[]{
        2.3,
        3.1,
        1.58
    };
    private String color;
    private double weight;
    // 继承Stub，也就是实现了ICat接口，并实现了IBinder接口
    public class CatBinder extends Stub
    {
        @Override
        public String getColor() throws RemoteException
        {
            return color;
        }
        @Override
        public double getWeight() throws RemoteException
        {
            return weight;
        }
    }
    @Override
    public void onCreate()
    {
        super.onCreate();
        catBinder = new CatBinder();
        timer.schedule(new TimerTask()
        {
            @Override
            public void run()
            {
                // 随机地改变Service组件内color、weight属性的值
                int rand = (int)(Math.random() * 3);
```

```
                    color = colors[rand];
                    weight = weights[rand];
                    System.out.println("--------" + rand);
                }
            } , 0 , 800);
        }
        @Override
        public IBinder onBind(Intent arg0)
        {
            /* 返回 catBinder 对象
             * 在绑定本地 Service 的情况下，该 catBinder 对象会直接
             * 传给客户端的 ServiceConnection 对象
             * 的 onServiceConnected 方法的第二个参数；
             * 在绑定远程 Service 的情况下，只将 catBinder 对象的代理
             * 传给客户端的 ServiceConnection 对象
             * 的 onServiceConnected 方法的第二个参数；
             */
            return catBinder; //①
        }
        @Override
        public void onDestroy()
        {
            timer.cancel();
        }
}
```

上面的程序中粗体字代码定义了一个 CatBinder 类，这个 CatBinder 类继承了 ICat.Stub 类，就是实现了 ICat 接口和 IBinder 接口，所以程序重写 onBind()方法时返回了该 CatBinder 的实例——如程序中①号粗体字代码所示。

通过上面的介绍不难看出，开发 AIDL 远程 Service 其实也很简单，只是需要比开发本地 Service 多定义一个 AIDL 接口而已。

该 Service 类开发完成之后，接下来还需要在 AndroidManifest.xml 文件中配置该 Service，也就是在 AndroidManifest.xml 文件中增加如下配置片段：

```
<!-- 定义一个 Service 组件 -->
<service android:name=".AidlService" >
    <intent-filter>
        <action android:name="org.crazyit.aidl.action.AIDL_SERVICE" />
    </intent-filter>
</service>
```

将该应用部署到手机（或模拟器）上，在程序列表中看不到这个应用——这是因为该应用并未提供 Activity 的缘故。但这没有关系，该应用所提供的 Service 可以供其他应用程序来调用。

➤➤10.2.4　客户端访问 AIDLService

正如前面提到的，AIDL 接口定义了两个进程之间的通信接口，因此不仅服务器端需要 AIDL 接口，客户端同样需要前面定义的 AIDL 接口，因此开发客户端的第一步就是将 Service 端的 AIDL 接口文件复制到客户端应用中，复制到客户端后 ADT 工具会为 AIDL 接口生成相应的实现。

客户端绑定远程 Service 与绑定本地 Service 的区别并不大，同样只需要两步。

- 创建 ServiceConnection 对象。

- 以 ServiceConnection 对象作为参数，调用 Context 的 bindService()方法绑定远程 Service 即可。

与绑定本地 Service 不同的是，绑定远程 Service 的 ServiceConnection 并不能直接获取 Service 的 onBind()方法所返回的对象，它只能返回 onBind()方法所返回的对象的代理，因此在 ServiceConnection 的 onServiceConnected 方法中需要通过如下代码进行处理：

```
catService = ICat.Stub.asInterface(service);
```

该示例的程序界面有一个按钮和两个文本框，当用户单击该按钮时将会访问远程 Service 的数据，两个文本框用于显示所获得数据。该程序的代码如下。

程序清单：codes\10\10.2\AidlClient\src\org\crazyit\client\AidlClient.java

```
public class AidlClient extends Activity
{
    private ICat catService;
    private Button get;
    EditText color, weight;
    private ServiceConnection conn = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName name
            , IBinder service)
        {
            // 获取远程 Service 的 onBind 方法返回的对象的代理
            catService = ICat.Stub.asInterface(service);
        }
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
            catService = null;
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        get = (Button) findViewById(R.id.get);
        color = (EditText) findViewById(R.id.color);
        weight = (EditText) findViewById(R.id.weight);
        // 创建所需绑定的 Service 的 Intent
        Intent intent = new Intent();
        intent.setAction("org.crazyit.aidl.action.AIDL_SERVICE");
        // 绑定远程 Service
        bindService(intent, conn, Service.BIND_AUTO_CREATE);
        get.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                try
                {
                    // 获取并显示远程 Service 的状态
                    color.setText(catService.getColor());
                    weight.setText(catService.getWeight() + "");
                }
                catch (RemoteException e)
                {
                    e.printStackTrace();
```

```
                    }
                }
            });
        }
        @Override
        public void onDestroy()
        {
            super.onDestroy();
            // 解除绑定
            this.unbindService(conn);
        }
}
```

将这个程序与前面绑定本地 Service 的代码进行对比，不难发现两个程序的差别很小，只是获取 Service 回调对象（IBinder 实例）的方式有所区别而已：绑定本地 Service 时可以直接获取 onBind 方法的返回值；绑定远程 Service 时获取的是 onBind 方法所返回对象的代理，因此需要进行一些处理，如上面的粗体字代码所示。



图 10.9　访问远程 Service 数据

运行该程序，单击程序界面中“获取远程 Service 的状态”按钮，将可以看到如图 10.9 所示的输出。

实例：传递复杂数据的 AIDL Service

本实例也是一个调用 AIDL Service 的例子，与前面的实例不同的是，该实例所传输的数据类型是自定义类型。

本例用到了两个自定义类型：Person 与 Pet，其中 Person 对象作为调用远程 Service 的参数，而 Pet 将作为返回值。就像 RMI 要求远程调用的参数和返回值都必须实现 Serializable 接口，Android 要求调用远程 Service 的参数和返回值都必须实现 Parcelable 接口。

实现 Parcelable 接口不仅要求实现该接口里定义的方法，而且要求在实现类中定义一个名为 CREATOR、类型为 Parcelable.Creator 的静态 Field。除此之外，还要求使用 AIDL 代码来定义这些自定义类型。



提示： 实现 Parcelable 接口相当于 Android 提供的一种自定义序列化机制。Java 序列化机制要求序列化类必须实现 Serializable 接口，而 Android 的序列化机制则要求自定义类必须实现 Parcelable 接口。

例如要定义 Person 类，先要 AIDL 来定义 Person 类，代码如下。

程序清单：codes\10\10.2\ComplexService\src\org\crazyit\service\Perosn.aidl

```
parcelable Person;
```

正如上面的代码所看到的，使用 AIDL 定义自定义类只要一行代码即可。

接下来定义一个实现 Parcelable 接口的 Person 类，该类代码如下。

程序清单：codes\10\10.2\ComplexService\src\org\crazyit\service\Person.java

```
public class Person implements Parcelable
{
    private Integer id;
    private String name;
    private String pass;
```

```
public Person()
{
}
public Person(Integer id, String name, String pass)
{
      super();
      this.id = id;
      this.name = name;
      this.pass = pass;
}
public Integer getId()
{
      return id;
}
public void setId(Integer id)
{
      this.id = id;
}
// 省略name、pass的setter和getter方法
...
@Override
public int hashCode()
{
      final int prime = 31;
      int result = 1;
      result = prime * result + ((name == null) ? 0 : name.hashCode());
      result = prime * result + ((pass == null) ? 0 : pass.hashCode());
      return result;
}
@Override
public boolean equals(Object obj)
{
      if (this == obj)
            return true;
      if (obj == null)
            return false;
      if (getClass() != obj.getClass())
            return false;
      Person other = (Person) obj;
      if (name == null)
      {
            if (other.name != null)
                  return false;
      }
      else if (!name.equals(other.name))
            return false;
      if (pass == null)
      {
            if (other.pass != null)
                  return false;
      }
      else if (!pass.equals(other.pass))
            return false;
      return true;
}
// 实现Parcelable接口必须实现的方法
@Override
public int describeContents()
{
      return 0;
}
```

```
    // 实现 Parcelable 接口必须实现的方法
    @Override
    public void writeToParcel(Parcel dest, int flags)
    {
        //把该对象所包含的数据写到 Parcel
        dest.writeInt(id);
        dest.writeString(name);
        dest.writeString(pass);
    }
    // 添加一个静态成员，名为 CREATOR，该对象实现了 Parcelable.Creator 接口
    public static final Parcelable.Creator<Person> CREATOR
        = new Parcelable.Creator<Person>() //①
    {
        @Override
        public Person createFromParcel(Parcel source)
        {
            // 从 Parcel 中读取数据，返回 Person 对象
            return new Person(source.readInt()
                , source.readString()
                , source.readString());
        }
        @Override
        public Person[] newArray(int size)
        {
            return new Person[size];
        }
    };
}
```

上面的程序定义了一个实现 Parcelable 接口的类，实现该接口主要就是要实现 writeToParcel(Parcel dest, int flags)方法，该方法负责把 Person 对象的数据写入 Parcel 中。与此同时，该类必须定义一个类型为 Parcelable.Creator<Person>、名为 CREATOR 的静态常量，该静态常量的值负责恢复从 Parcel 数据包中恢复 Person 对象，因此该对象定义的 createFromPerson()方法用于恢复 Person 对象。



提示：

实际上让 Person 类实现 Parcelable 接口也是一种序列化机制，只是 Android 没有直接使用 Java 提供的序列化机制，而是提供了 Parcelable 这种轻量级的序列化机制。

定义了 Person 自定义类之后，接下来还需要定义一个 Pet 类，定义 Pet 类的方式与定义 Person 类的方式大致相似，故此处不再给出详细代码，读者可以自行参考光盘代码。

有了 Person、Pet 自定义类之后，接下来就可以使用 AIDL 来定义通信接口了，定义通信接口的代码如下。

程序清单：codes\10\10.2\ComplexService\src\org\crazyit\service\IPet.aidl

```
interface IPet
{
    // 定义一个 Person 对象作为传入参数
    List<Pet> getPets(in Person owner);
}
```

正如上面的粗体字代码所看到的，在 AIDL 接口中定义方法时，需要指定形参的传递模式，对于 Java 语言来说，一般都是采用传入参数的方式，因此上面指定为 in 模式。

与前面介绍类似的是，当定义好这个 AIDL 接口之后，ADT 工具会自动为它生成相应的实现，这不需要开发者操心。接下来就是开发一个 Service 类了，让 Service 类的 onBind 方法返回 IPet 实现类的实例。该 Service 类的代码如下。

程序清单：codes\10\10.2\ComplexService\src\org\crazyit\service\ComplexService.java

```
public class ComplexService extends Service
{
    private PetBinder petBinder;
    private static Map<Person , List<Pet>> pets
        = new HashMap<Person , List<Pet>>();
    static
    {
        // 初始化 pets Map 集合
        ArrayList<Pet> list1 = new ArrayList<Pet>();
        list1.add(new Pet("旺财" , 4.3));
        list1.add(new Pet("来福" , 5.1));
        pets.put(new Person(1, "sun" , "sun") , list1);
        ArrayList<Pet> list2 = new ArrayList<Pet>();
        list2.add(new Pet("kitty" , 2.3));
        list2.add(new Pet("garfield" , 3.1));
        pets.put(new Person(2, "bai" , "bai") , list2);
    }
    // 继承 Stub，也就是实现了 IPet 接口，并实现了 IBinder 接口
    public class PetBinder extends Stub
    {
        @Override
        public List<Pet> getPets(Person owner) throws RemoteException
        {
            // 返回 Service 内部的数据
            return pets.get(owner);
        }
    }
    @Override
    public void onCreate()
    {
        super.onCreate();
        petBinder = new PetBinder();
    }
    @Override
    public IBinder onBind(Intent arg0)
    {
        /* 返回 catBinder 对象
         * 在绑定本地 Service 的情况下，该 catBinder 对象会直接
         * 传给客户端的 ServiceConnection 对象
         * 的 onServiceConnected 方法的第二个参数；
         * 在绑定远程 Service 的情况下，只将 catBinder 对象的代理
         * 传给客户端的 ServiceConnection 对象
         * 的 onServiceConnected 方法的第二个参数；
         */
        return petBinder; //①
    }
    @Override
    public void onDestroy()
    {
    }
}
```

与前面介绍的 AIDL Service 类一样，这个 Service 类也实现了 onBind()方法，并让该方法返回了 IPet.Stub 的子类，该 Service 类开发完成。

在 AndroidManifest.xml 文件中配置该 Service，配置 Service 的代码无须任何改变，此处不再给出。

开发客户端的第一步不仅需要把 IPet.aidl 文件复制过去，还需要把定义 Person 类的 Java 文件、AIDL 文件、定义 Pet 类的 Java 文件、AIDL 文件也复制过去。

客户端依然按之前的方式来绑定远程 Service 即可，并在 ServiceConnection 实现类的 onServiceConnected()方法中获取远程 Service 的 onBind()方法返回的代理对象即可。该程序使用 ListView 来显示远程 Service 所返回的 Pet 集合，客户端程序代码如下。

程序清单：codes\10\10.2\ComplexClient\src\org\crazyit\client\ComplexClient.java

```
public class ComplexClient extends Activity
{
    private IPet petService;
    private Button get;
    EditText personView;
    ListView showView;
    private ServiceConnection conn = new ServiceConnection()
    {
        @Override
        public void onServiceConnected(ComponentName name
            , IBinder service)
        {
            // 获取远程 Service 的 onBind 方法返回的对象的代理
            petService = IPet.Stub.asInterface(service);
        }
        @Override
        public void onServiceDisconnected(ComponentName name)
        {
            petService = null;
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        personView = (EditText) findViewById(R.id.person);
        showView = (ListView) findViewById(R.id.show);
        get = (Button) findViewById(R.id.get);
        // 创建所需绑定的 Service 的 Intent
        Intent intent = new Intent();
        intent.setAction("org.crazyit.aidl.action.COMPLEX_SERVICE");
        // 绑定远程 Service
        bindService(intent, conn, Service.BIND_AUTO_CREATE);
        get.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                try
                {
                    String personName = personView.getText().toString();
                    // 调用远程 Service 的方法
                    List<Pet> pets = petService.getPets(new Person(1,
                        personName, personName)); //①
                    // 将程序返回的 List 包装成 ArrayAdapter
                    ArrayAdapter<Pet> adapter = new ArrayAdapter<Pet>(
                        ComplexClient.this,
                        android.R.layout.simple_list_item_1, pets);
```

```
                        showView.setAdapter(adapter);
                    }
                    catch (RemoteException e)
                    {
                        e.printStackTrace();
                    }
                }
            });
        }
        @Override
        public void onDestroy()
        {
            super.onDestroy();
            // 解除绑定
            this.unbindService(conn);
        }
}
```

与前面介绍的绑定远程 Service 一样，本程序同样也是先定义一个 ServiceConnection 对象，并在该对象的 onServiceConnected 方法中获得了远程 Service 的 onBind 方法所返回对象的代理。

当用户单击程序中的按钮时，程序将以用户在文本框中输入的字符串来构建 Person 对象，并作为参数调用 petService 对象（远程 Service 的回调对象）的方法，从而可以获取远程 Service 的内部状态。运行该程序，将看到如图 10.10 所示的界面。

除了这些由用户自行开发、启动的 Service 之外，Android 系统本身提供了大量的系统 Service，开发者只要在程序中调用 Context 的如下方法即可获取这些系统 Service。

- getSystemService(String name)：根据 Service 名称来获取系统 Service。

一旦获取了这些 Service，接下来就可以在程序中进行相应的操作了，接下来详细介绍 Android 所提供的这些常见的相关操作。



图 10.10 传递复杂数据的远程 Service

10.3 电话管理器（TelephonyManager）

TelephonyManager 是一个管理手机通话状态、电话网络信息的服务类，该类提供了大量的 getXxx()方法来获取电话网络的相关信息。

在程序中获取 TelephonyManager 十分简单，只要调用如下代码即可：

```
TelephonyManager tManager =
 (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

接下来就可以通过 TelephonyManager 获取相关信息或者进行相关操作了。

实例：获取网络和 SIM 卡信息

通过 TelephonyManager 提供的系列方法即可获取手机网络、SIM 卡的相关信息，该程序使用了一个 ListView 来显示网路和 SIM 卡的相关信息。

该程序代码如下。

程序清单：codes\10\10.3\TelephonyStatus\src\org\crazyit\manager\TelephonyStatus.java

```
public class TelephonyStatus extends Activity
{
```

```
ListView showView;
// 声明代表状态名的数组
String[] statusNames;
// 声明代表手机状态的集合
ArrayList<String> statusValues = new ArrayList<String>();
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // 获取系统的 TelephonyManager 对象
    TelephonyManager tManager = (TelephonyManager)
        getSystemService(Context.TELEPHONY_SERVICE);
    // 获取各种状态名称的数组
    statusNames = getResources().getStringArray(R.array.statusNames);
    // 获取代表 SIM 卡状态的数组
    String[] simState = getResources()
        .getStringArray(R.array.simState);
    // 获取代表电话网络类型的数组
    String[] phoneType = getResources().getStringArray(
        R.array.phoneType);
    // 获取设备编号
    statusValues.add(tManager.getDeviceId());
    // 获取系统平台的版本
    statusValues.add(tManager.getDeviceSoftwareVersion()
        != null ? tManager.getDeviceSoftwareVersion() : "未知");
    // 获取网络运营商代号
    statusValues.add(tManager.getNetworkOperator());
    // 获取网络运营商名称
    statusValues.add(tManager.getNetworkOperatorName());
    // 获取手机网络类型
    statusValues.add(phoneType[tManager.getPhoneType()]);
    // 获取设备所在位置
    statusValues.add(tManager.getCellLocation() != null ? tManager
        .getCellLocation().toString() : "未知位置");
    // 获取 SIM 卡的国别
    statusValues.add(tManager.getSimCountryIso());
    // 获取 SIM 卡序列号
    statusValues.add(tManager.getSimSerialNumber());
    // 获取 SIM 卡状态
    statusValues.add(simState[tManager.getSimState()]);
    // 获得 ListView 对象
    showView = (ListView) findViewById(R.id.show);
    ArrayList<Map<String, String>> status =
        new ArrayList<Map<String, String>>();
    // 遍历 statusValues 集合，将 statusNames、statusValues
    // 的数据封装到 List<Map<String , String>>集合中
    for (int i = 0; i < statusValues.size(); i++)
    {
        HashMap<String, String> map = new HashMap<String, String>();
        map.put("name", statusNames[i]);
        map.put("value", statusValues.get(i));
        status.add(map);
    }
    // 使用 SimpleAdapter 封装 List 数据
    SimpleAdapter adapter = new SimpleAdapter(this, status,
        R.layout.line, new String[] { "name", "value" }
        , new int[] { R.id.name, R.id.value });
    // 为 ListView 设置 Adapter
    showView.setAdapter(adapter);
}
}
```

运行该程序，可以看到程序有如图 10.11 所示的输出。

TelephonyManager 除了提供系列的 getXxx()方法来获取网络状态和 SIM 卡信息之外，还提供了一个 listen(PhoneStateListener listener, int events)方法来监听通话状态。下面通过该方法来监听手机来电信息。



图 10.11 获取 SIM 卡和网络状态

实例：监听手机来电

下面的程序监听 TelephonyManager 的通话状态来监听手机的所有来电，该程序代码如下。

程序清单：codes\10\10.3\MonitorPhone\src\org\crazyit\manager\MonitorPhone.java

```
public class MonitorPhone extends Activity
{
    TelephonyManager tManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 取得 TelephonyManager 对象
        tManager = (TelephonyManager)
            getSystemService(Context.TELEPHONY_SERVICE);
        // 创建一个通话状态监听器
        PhoneStateListener listener = new PhoneStateListener()
        {
            @Override
            public void onCallStateChanged(int state, String number)
            {
                switch (state)
                {
                    // 无任何状态
                    case TelephonyManager.CALL_STATE_IDLE:
                        break;
                    case TelephonyManager.CALL_STATE_OFFHOOK:
                        break;
                    // 来电铃响时
                    case TelephonyManager.CALL_STATE_RINGING:
                        OutputStream os = null;
                        try
                        {
                            os = openFileOutput("phoneList", MODE_APPEND);
                        }
                        catch (FileNotFoundException e)
                        {
                            e.printStackTrace();
                        }
                        PrintStream ps = new PrintStream(os);
                        // 将来电号码记录到文件中
                        ps.println(new Date() + " 来电：" + number);
                        ps.close();
                        break;
                    default:
                        break;
                }
                super.onCallStateChanged(state, number);
            }
```

```
            };
            // 监听电话通话状态的改变
            tManager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
        }
    }
```

上面的程序中首先创建了一个 PhoneStateListener，它是一个通话状态监听器，该监听器可用于对 TelephonyManager 进行监听。当手机来电铃响时，程序将会把来电号码记录到文件中，如上面的粗体字代码所示。

运行上面的程序，在保证该程序运行的状态下，启动另一个模拟器呼叫该电话。接下来就可以在 DDMS 的 File Explorer 面板的 data/data/org.crazyit.manager/files 目录下看到一个 phoneList 文件，将该文件导出到计算机上，查看该文件内容即可看到如下文件内容：

```
Sun Nov 11 15:19:38 GMT 2012 来电：15555215556
```

从上面文件内容可以看出，该程序记录了来自另一个模拟器的电话呼入。

如果我们把这段代码放在后台执行的 Service 中运行，并且设置 Serivce 组件随着系统开机自动运行，那么这种监听就可以“神不知鬼不觉”了。本章后面会介绍如何让 Service 随系统开机自动运行。

需要指出的是，由于该程序需要获取手机的通话状态，因此必须在 AndroidManifest.xml 文件中增加如下权限配置代码：

```
<!-- 授予该应用读取通话状态的权限 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

实例：黑名单来电自动挂断

本示例允许用户动态加载手机通信录中的电话号码，勾选某些号码作为黑名单，一旦将某些号码设为黑名单，当该号码呼入时，系统将会自动挂断该号码。

与前一个示例相似的是，程序同样也是监听 TelephonyManager 的通话状态来实现该功能，当系统检测到电话呼入时，立即判断该号码是否在黑名单中，如果该号码在黑名单中立即挂断该号码即可。

不幸的是，Android 没有对外公开挂断电话的 API，如果需要挂断电话，必须使用 AIDL 与电话管理 Service 进行通信，并调用服务中的 API 实现结束通话。

为了调用远程的 AIDL Service，开发者需要将 Android 源代码中如下两个文件复制到项目的相应位置：

- com.android.internal.telephony 包下的 ITelephony.aidl。
- android.telephony 包下的 NeighboringCellInfo.aidl。

开发者需要在项目中建立对应的包，然后将这两个文件复制到相应的包下。一旦将这两个*.aidl 文件复制到项目的 src 目录的相应的包下，ADT 会在根目录下自动生成 ITelephony.java 源文件。

接下来就可以在程序中调用 ITelephony 的 endCall 方法来挂断电话了。

本程序还调用了系统联系人的 ContentProvider 来获取系统联系人信息，获取系统联系人信息之后，程序提供了一个带复选框的列表供用户勾选黑名单。

该程序的界面很简单，只是提供一个按钮让用户打开列表对话框来勾选黑名单，故不再给出界面布局文件。该程序代码如下。

程序清单：codes\10\10.3\BlockList\src\org\crazyit\manager\BlockMain.java

```
public class BlockMain extends Activity
{
    // 记录黑名单的List
    ArrayList<String> blockList = new ArrayList<String>();
    TelephonyManager tManager;
    // 监听通话状态的监听器
    CustomPhoneCallListener cpListener;
    public class CustomPhoneCallListener extends PhoneStateListener
    {
        @Override
        public void onCallStateChanged(int state, String number)
        {
            switch (state)
            {
                case TelephonyManager.CALL_STATE_IDLE:
                    break;
                case TelephonyManager.CALL_STATE_OFFHOOK:
                    break;
                // 当电话呼入时
                case TelephonyManager.CALL_STATE_RINGING:
                    // 如果该号码属于黑名单
                    if (isBlock(number))
                    {
                        try
                        {
                            Method method = Class.forName(
                                "android.os.ServiceManager")
                                .getMethod("getService"
                                , String.class);
                            // 获取远程TELEPHONY_SERVICE的IBinder对象的代理
                            IBinder binder = (IBinder) method.invoke(null,
                                new Object[] { TELEPHONY_SERVICE });
                            // 将IBinder对象的代理转换为ITelephony对象
                            ITelephony telephony = ITelephony.Stub
                                .asInterface(binder);
                            // 挂断电话
                            telephony.endCall();
                        }
                        catch (Exception e)
                        {
                            e.printStackTrace();
                        }
                    }
                    break;
            }
            super.onCallStateChanged(state, number);
        }
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的TelephonyManager管理器
        tManager = (TelephonyManager)
            getSystemService(TELEPHONY_SERVICE);
        cpListener = new CustomPhoneCallListener();
        // 通过TelephonyManager监听通话状态的改变
        tManager.listen(cpListener
```

```
        , PhoneStateListener.LISTEN_CALL_STATE);
    // 获取程序的按钮，并为它的单击事件绑定监听器
    findViewById(R.id.managerBlock).setOnClickListener(
        new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 查询联系人的电话号码
                final Cursor cursor = getContentResolver()
                    .query(ContactsContract.CommonDataKinds
                    .Phone.CONTENT_URI,      null, null, null, null);
                BaseAdapter adapter = new BaseAdapter()
                {
                    @Override
                    public int getCount()
                    {
                        return cursor.getCount();
                    }
                    @Override
                    public Object getItem(int position)
                    {
                        return position;
                    }
                    @Override
                    public long getItemId(int position)
                    {
                        return position;
                    }
                    @Override
                    public View getView(int position,
                        View convertView, ViewGroup parent)
                    {
                        cursor.moveToPosition(position);
                        CheckBox rb = new CheckBox(BlockMain.this);
                        // 获取联系人的电话号码，并去掉中间的中画线
                        String number = cursor
                            .getString(cursor.getColumnIndex(
                            ContactsContract.CommonDataKinds
                            .Phone.NUMBER))
                            .replace("-", "");
                        rb.setText(number);
                        // 如果该号码已经被加入黑名单，默认勾选该号码
                        if (isBlock(number))
                        {
                            rb.setChecked(true);
                        }
                        return rb;
                    }
                };
                // 加载 list.xml 布局文件对应的 View
                View selectView = getLayoutInflater().inflate(
                    R.layout.list, null);
                // 获取 selectView 中的名为 list 的 ListView 组件
                final ListView listView = (ListView) selectView
                    .findViewById(R.id.list);
                listView.setAdapter(adapter);
                new AlertDialog.Builder(BlockMain.this)
                    .setView(selectView)
                    .setPositiveButton("确定",
                    new DialogInterface.OnClickListener()
```

```
                {
                    @Override
                    public void onClick(
                        DialogInterface dialog, int which)
                    {
                        // 清空 blockList 集合
                        blockList.clear();
                        // 遍历 listView 组件的每个列表项
                        for (int i = 0; i < listView
                                .getCount(); i++)
                        {
                            CheckBox checkBox = (CheckBox)
                                listView.getChildAt(i);
                            // 如果该列表项被勾选
                            if (checkBox.isChecked())
                            {
                                // 添加该列表项的电话号码
                                blockList.add(checkBox
                                    .getText().toString());
                            }
                        }
                        System.out.println(blockList);
                    }
                }).show();
            }
        });
    }
    // 判断某个电话号码是否在黑名单之内
    public boolean isBlock(String phone)
    {
        for (String s1 : blockList)
        {
            if (s1.equals(phone))
            {
                return true;
            }
        }
        return false;
    }
}
```

当有电话呼入时，如果呼入号码在黑名单内，程序就会自动挂断该电话，如程序中粗体字代码所示。程序中还提供了一个对话框让用户勾选黑名单。运行该程序，单击“管理黑名单”按钮，系统打开如图 10.12 所示的对话框。

图 10.12 所示对话框中将 15555215556 号码勾选成黑名单号码，这个号码是另一台模拟器的号码。保持该程序处于运行状态，通过另一个模拟器向当前程序运行的模拟器呼入电话，将可以看到另一台模拟器开始呼入后立即显示如图 10.13 所示的界面。



图 10.12 勾选黑名单



图 10.13 被对方设置的黑名单自动挂断

这个程序已经是一个非常实用的应用了，如果用户不希望接听某个号码的来电，只要把此人的号码添加到黑名单中即可。当然，这个程序还可进行如下三点改进：

- 程序可通过后台运行的 Service 来监听号码，并设置该 Service 随系统开机自动运行。
- 程序应该把黑名单（也就是程序中 blockList 集合里的元素）写入文件中，这样即使手机关机、程序退出，黑名单信息依然不会丢失。
- 程序还应该提供一个输入框，让用户自行输入需要屏蔽的电话号码。这样该程序不仅可以屏蔽系统联系人的号码，也可屏蔽任何想屏蔽的号码。

这三点都很容易实现，相信读者有能力改进并实现这些功能。

需要指出的是，由于该程序需要获取手机的通话状态并需要控制手机通话状态，还需要读取联系人信息，因此必须在 AndroidManifest.xml 文件中增加如下权限配置代码：

```
<!-- 授予该应用控制通话的权限 -->
<uses-permission android:name="android.permission.CALL_PHONE" />
<!-- 授予该应用读取通话状态的权限 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<!-- 授予读联系人 ContentProvider 的权限 -->
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

10.4　短信管理器（SmsManager）

SmsManager 是 Android 提供的另一个非常常见的服务，SmsManager 提供了系列 sendXxxMessage()方法用于发送短信，不过就现在实际应用来看，短信通常都是普通的文本内容，也就是调用 sendTextMessage()方法进行发送即可。

实例：发送短信

下面的程序十分简单，程序提供一个文本框让用户输入收件人号码，一个文本框让用户输入短信内容，接下来单击“发送”按钮即可将短信发送出去。

程序代码如下。

程序清单：codes\10\10.4\SendSms\src\org\crazyit\manager\SendSms.java

```
public class SendSms extends Activity
{
    EditText number, content;
    Button send;
    SmsManager sManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取 SmsManager
        sManager = SmsManager.getDefault();
        // 获取程序界面上的两个文本框和按钮
        number = (EditText) findViewById(R.id.number);
        content = (EditText) findViewById(R.id.content);
        send = (Button) findViewById(R.id.send);
        // 为 send 按钮的单击事件绑定监听器
        send.setOnClickListener(new OnClickListener()
        {
```

```
            @Override
            public void onClick(View arg0)
            {
                // 创建一个 PendingIntent 对象
                PendingIntent pi = PendingIntent.getActivity(
                    SendSms.this, 0, new Intent(), 0);
                // 发送短信
                sManager.sendTextMessage(number.getText().toString(),
                    null, content.getText().toString(), pi, null);
                // 提示短信发送完成
                Toast.makeText(SendSms.this, "短信发送完成", 8000).show();
            }
        });
    }
}
```

从上面程序的粗体字代码可以看出，使用 SmsManager 发送短信十分简单，简单地调用 sendTextMessage()方法即可发送。

上面的程序中用到了一个 PendingIntent 对象，PendingIntent 是对 Intent 的包装，一般通过调用 PendingIntent 的 getActivity()、getService()、getBroadcastReceiver()静态方法来获取 PendingIntent 对象。与 Intent 对象不同的是：PendingIntent 通常会传给其他应用组件，从而由其他应用程序来执行 PendingIntent 所包装的“Intent”。

该程序需要调用 SmsManager 来发送短信，因此还需要授予该程序发送短信的权限，也就是在 AndroidManifest.xml 文件中增加如下代码：

```
<!-- 授予发送短信的权限 -->
<uses-permission android:name="android.permission.SEND_SMS"/>
```

实例：短信群发

短信群发也是一个十分实用的功能，逢年过节，很多人都喜欢通过短信群发向自己的朋友表示祝福。短信群发可以将一条短信同时向多个人发送。短信群发的实现十分简单，只要让程序遍历每个收件人号码并依次向每个收件人发送短信即可。

该程序也提供了一个带列表框的对话框供用户选择收件人号码，代码如下。

程序清单：codes\10\10.4\GroupSend\src\org\crazyit\manager\GroupSend.java

```
public class GroupSend extends Activity
{
    EditText numbers, content;
    Button select, send;
    SmsManager sManager;
    // 记录需要群发的号码列表
    ArrayList<String> sendList = new ArrayList<String>();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        sManager = SmsManager.getDefault();
        // 获取界面上的文本框、按钮组件
        numbers = (EditText) findViewById(R.id.numbers);
        content = (EditText) findViewById(R.id.content);
        select = (Button) findViewById(R.id.select);
        send = (Button) findViewById(R.id.send);
        // 为 send 按钮的单击事件绑定监听器
        send.setOnClickListener(new OnClickListener()
```

```
{
    @Override
    public void onClick(View v)
    {
        for (String number : sendList)
        {
            // 创建一个 PendingIntent 对象
            PendingIntent pi = PendingIntent.getActivity(
                GroupSend.this, 0, new Intent(), 0);
            // 发送短信
            sManager.sendTextMessage(number, null, content
                .getText().toString(), pi, null);
        }
        // 提示短信群发完成
        Toast.makeText(GroupSend.this, "短信群发完成"
            , Toast.LENGTH_SHORT).show();
    }
});
// 为 select 按钮的单击事件绑定监听器
select.setOnClickListener(new OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        // 查询联系人的电话号码
        final Cursor cursor = getContentResolver().query(
            ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
            null, null, null, null);
        BaseAdapter adapter = new BaseAdapter()
        {
            @Override
            public int getCount()
            {
                return cursor.getCount();
            }
            @Override
            public Object getItem(int position)
            {
                return position;
            }
            @Override
            public long getItemId(int position)
            {
                return position;
            }
            @Override
            public View getView(int position, View convertView,
                ViewGroup parent)
            {
                cursor.moveToPosition(position);
                CheckBox rb = new CheckBox(GroupSend.this);
                // 获取联系人的电话号码，并去掉中间的中画线、空格
                String number = cursor
                    .getString(cursor.getColumnIndex(ContactsContract
                    .CommonDataKinds.Phone.NUMBER))
                    .replace("-", "")
                    .replace(" " , "");
                rb.setText(number);
                // 如果该号码已经被加入发送人名单，默认勾选该号码
                if (isChecked(number))
                {
```

```
                        rb.setChecked(true);
                    }
                    return rb;
                }
            };
            // 加载 list.xml 布局文件对应的 View
            View selectView = getLayoutInflater().inflate(
                R.layout.list, null);
            // 获取 selectView 中的名为 list 的 ListView 组件
            final ListView listView = (ListView) selectView
                .findViewById(R.id.list);
            listView.setAdapter(adapter);
            new AlertDialog.Builder(GroupSend.this)
                .setView(selectView)
                .setPositiveButton("确定",
                new DialogInterface.OnClickListener()
                {
                    @Override
                    public void onClick(DialogInterface dialog,
                        int which)
                    {
                        // 清空 sendList 集合
                        sendList.clear();
                        // 遍历 listView 组件的每个列表项
                        for (int i = 0; i < listView.getCount(); i++)
                        {
                            CheckBox checkBox = (CheckBox) listView
                                .getChildAt(i);
                            // 如果该列表项被勾选
                            if (checkBox.isChecked())
                            {
                                // 添加该列表项的电话号码
                                sendList.add(checkBox.getText()
                                    .toString());
                            }
                        }
                        numbers.setText(sendList.toString());
                    }
                }).show();
        }
    });
}
// 判断某个电话号码是否已在群发范围内
public boolean isChecked(String phone)
{
    for (String s1 : sendList)
    {
        if (s1.equals(phone))
        {
            return true;
        }
    }
    return false;
}
}
```

该程序的实现也很简单，程序提供了一个带列表的对话框供用户选择群发短信的收件人号码，程序则使用了一个 ArrayLsit<String>集合来保存所有的收件人号码。为了实现群发功能，程序使用循环遍历 ArrayList<String>中的每个号码，并使用 SmsManager 依次向每个号码发送短信即可，如上面的粗体字代码所示。



还有一点需要指出，该程序有一个潜在的风险：该程序直接在主线程中采用循环向多个人发送短信，如果需要发送短信的人太多且网络延迟严重，群发短信就会变成一个耗时任务，此时可以考虑使用 IntentService 来群发短信，群发完成后通过广播通知前台 Activity。

10.5 音频管理器（AudioManager）

在某些时候，程序需要管理系统音量，或者直接让系统静音，这就可借助于 Android 提供的 AudioManager 来实现。程序一样是调用 getSystemService()方法来获取系统的音频管理器。接下来就可调用 AudioManager 的方法来控制手机音频了。

10.5.1 AudioManager 简介

在程序获取了 AudioManager 对象之后，接下来就可调用 AudioManager 的如下常用方法来控制手机音频了。

- adjustStreamVolume(int streamType, int direction, int flags)：调整手机指定类型的声音。其中第一个参数 streamType 指定声音类型，该参数可接受如下几个值。
- STREAM_ALARM：手机闹铃的声音。
- STREAM_DTMF：DTMF 音调的声音。
- STREAM_MUSIC：手机音乐的声音。
- STREAM_NOTIFICATION：系统提示的声音。
- STREAM_RING：电话铃声的声音。
- STREAM_SYSTEM：手机系统的声音。
- STREAM_VOICE_CALL：语音电话的声音。

第二个参数指定对声音进行增大、还是减少；第三个参数是调整声音时的标志，例如指定 FLAG_SHOW_UI，则指定调整声音时显示音量进度条。

- setMicrophoneMute(boolean on)：设置是否让麦克风静音。
- setMode(int mode)：设置声音模式，可设置的值有 NORMAL、RINGTONE 和 IN_CALL。
- setRingerMode(int ringerMode)：设置手机的电话铃声的模式。可支持如下几个属性值。
- RINGER_MODE_NORMAL：正常的手机铃声。
- RINGER_MODE_SILENT：手机铃声静音。
- RINGER_MODE_VIBRATE：手机振动。
- setSpeakerphoneOn(boolean on)：设置是否打开扩音器。
- setStreamMute(int streamType, boolean state)：将手机的指定类型的声音调整为静音。其中 streamType 参数与 adjustStreamVolume 方法中第一个参数的意义相同。

- setStreamVolume(int streamType, int index, int flags)：直接设置手机的指定类型的音量值。其中 streamType 参数与 adjustStreamVolume 方法中第一个参数的意义相同。

下面将通过一个简单的示例来示范 AudioManager 控制手机音频。

实例：使用 AudioManager 控制手机音频

本程序中提供一个按钮用于播放音乐，系统使用 MediaPlayer 播放音乐，程序还提供两个按钮控制音乐声的音量的提高、降低，并使用一个 ToggleButton 来控制是否静音。

该程序的界面比较简单，只是包含几个简单的按钮即可，该程序代码如下。

程序清单：codes\10\10.5\AudioTest\src\org\crazyit\manager\AudioTest.java

```
public class AudioTest extends Activity
{
    Button play, up, down;
    ToggleButton mute;
    AudioManager aManager;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的音频服务
        aManager = (AudioManager) getSystemService(
            Service.AUDIO_SERVICE);
        // 获取界面中三个按钮和一个 ToggleButton 控件
        play = (Button) findViewById(R.id.play);
        up = (Button) findViewById(R.id.up);
        down = (Button) findViewById(R.id.down);
        mute = (ToggleButton) findViewById(R.id.mute);
        // 为 play 按钮的单击事件绑定监听器
        play.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 初始化 MediaPlayer 对象，准备播放音乐
                MediaPlayer mPlayer = MediaPlayer.create(
                    AudioTest.this, R.raw.earth);
                // 设置循环播放
                mPlayer.setLooping(true);
                // 开始播放
                mPlayer.start();
            }
        });
        up.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 指定调节音乐的音频，增大音量，而且显示音量图形示意
                aManager.adjustStreamVolume(AudioManager.STREAM_MUSIC,
                    AudioManager.ADJUST_RAISE, AudioManager.FLAG_SHOW_UI);
            }
        });
        down.setOnClickListener(new OnClickListener()
        {
```

```
                @Override
                public void onClick(View source)
                {
                    // 指定调节音乐的音频，降低音量，而且显示音量图形示意
                    aManager.adjustStreamVolume(AudioManager.STREAM_MUSIC,
                        AudioManager.ADJUST_LOWER, AudioManager.FLAG_SHOW_UI);
                }
            });
            mute.setOnCheckedChangeListener(new OnCheckedChangeListener()
            {
                @Override
                public void onCheckedChanged(CompoundButton source,
                    boolean isChecked)
                {
                    // 指定调节音乐的音频，根据 isChecked 确定是否需要静音
                    aManager.setStreamMute(AudioManager.STREAM_MUSIC,
                        isChecked);
                }
            });
        }
    }
```



图 10.14　使用 AudioManager 管理音频

上面的程序中第一段粗体字代码使用 AudioManager 的 adjustStreamVolume 提高播放音乐的音量；第二段粗体字代码使用 AudioManager 的 adjustStreamVolume 降低播放音乐的音量；第三段粗体字代码则调用 AudioManager 的 setStreamMute()方法将音乐设为静音。

运行该程序，单击程序中“增大音量”或“降低音量”按钮，系统播放音乐音量将会随之改变，如图 10.14 所示。

10.6　振动器（Vibrator）

在某些时候，程序需要启动系统振动器，比如手机静音时使用振动提示用户；再比如玩游戏时，当系统碰撞、爆炸时使用振动带给用户更逼真的体验等。总之，振动是除视频、声音之外的另一种“多媒体”，充分利用系统的振动器会带给用户更好的体验。

系统获取 Vibrator 也是调用 Context 的 getSystemService()方法即可，接下来就可调用 Vibrator 的方法来控制手机振动了。

➤➤10.6.1　Vibrator 简介

Vibrator 的使用比较简单，它只有三个简单的方法来控制手机振动。

- vibrate(long milliseconds)：控制手机振动 milliseconds 毫秒。
- vibrate(long[] pattern, int repeat)：指定手机以 pattern 指定的模式振动。例如指定 pattern 为 new int[400 , 800 , 1200, 1600]，就是指定在 400ms、800ms、1200ms、1600ms 这些时间点交替启动、关闭手机振动器；其中 repeat 指定 pattern 数组的索引，指定对 pattern 数组中从 repeat 索引开始的振动进行循环。
- cancel()：关闭手机振动。

掌握这些方法之后，接下来就可在程序中通过 Vibrator 来控制手机振动了。

10.6.2 使用 Vibrator 控制手机振动

下面这个程序十分简单，程序几乎不需要界面，重写了 Activity 的 onTouchEvent (MotionEvent event)方法，这样使得用户触碰手机触摸屏时将会启动手机振动。程序代码如下。

程序清单：codes\10\10.6\VibratorTest\src\org\crazyit\manager\VibratorTest.java

```
public class VibratorTest extends Activity
{
    Vibrator vibrator;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取系统的 Vibrator 服务
        vibrator = (Vibrator) getSystemService(
            Service.VIBRATOR_SERVICE);
    }
    // 重写 onTouchEvent 方法，当用户触碰触摸屏时触发该方法
    @Override
    public boolean onTouchEvent(MotionEvent event)
    {
        Toast.makeText(this, "手机振动"
            , Toast.LENGTH_LONG).show();
        // 控制手机振动 2 秒
        vibrator.vibrate(2000);
        return super.onTouchEvent(event);
    }
}
```

上面的程序中第一行粗体字代码用于获取系统的 Vibrator 对象，第二行粗体字代码用于开始手机振动 2 秒。

需要指出的是，程序控制手机振动需要得到相应的权限，因此不要忘了在 AndroidManifest.xml 文件中增加如下授权配置代码：

```
<!-- 授予程序访问振动器的权限 -->
<uses-permission android:name="android.permission.VIBRATE"/>
```

在模拟器中运行该程序看不出振动效果，建议读者将该程序部署到真机上运行该程序。

10.7 手机闹钟服务（AlarmManager）

AlarmManager 通常的用途就是用来开发手机闹钟，但实际上它的作用不止于此。它的本质是一个全局的定时器，AlarmManager 可在指定时间或指定周期启动其他组件（包括 Activity、Service、BroadcastReceiver）。

10.7.1 AlarmManager 简介

AlarmManager 不仅可用于开发闹钟应用，还可作为一个全局定时器使用，Android 应用的程序中也是通过 Context 的 getSystemService()方法来获取 AlarmManager 对象，一旦程序获取了 AlarmManager 对象之后，就可调用它的如下方法来设置定时启动指定组件。

- set(int type, long triggerAtTime, PendingIntent operation)：设置在 triggerAtTime

时间启动由 operation 参数指定的组件。其中第一个参数指定定时服务的类型，该参数可接受如下值。

- ELAPSED_REALTIME：指定从现在开始时间过了一定时间后启动 operation 所对应的组件。
- ELAPSED_REALTIME_WAKEUP：指定从现在开始时间过了一定时间后启动 operation 所对应的组件。即使系统关机也会执行 operation 所对应的组件。
- RTC：指定当系统调用 System.currentTimeMillis()方法返回值与 triggerAtTime 相等时启动 operation 所对应的组件。
- RTC_WAKEUP：指定当系统调用 System.currentTimeMillis()方法返回值与 triggerAtTime 相等时启动 operation 所对应的组件。即使系统关机也会执行 operation 所对应的组件。
- setInexactRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)：设置一个非精确的周期性任务。例如我们设置 Alarm 每个小时启动一次，但系统并不一定总在每个小时的开始启动 Alarm 服务。
- setRepeating(int type, long triggerAtTime, long interval, PendingIntent operation)：设置一个周期性执行的定时服务。
- cancel(PendingIntent operation)：取消 AlarmManager 的定时服务。

掌握了 AlarmManager 的如上功能之后，接下来我们通过两个示例来示范 AlarmManager 在实际开发中的用途。

10.7.2 设置闹钟

这个程序比较简单，程序提供一个按钮让用户来设置闹铃时间（单击该按钮将会打开一个时间设置对话框），当用户设置好闹铃时间之后，即使退出该程序，到了预设时间 ManagerAlarm 一样会启动指定组件——这是因为 AlarmManager 是一个全局定时器的缘故。

该程序的界面布局很简单，程序界面上只有一个简单的按钮，程序代码如下。

程序清单：codes\10\10.7\AlarmTest\src\org\crazyit\manager\AlarmTest.java

```
public class AlarmTest extends Activity
{
    Button setTime;
    AlarmManager aManager;
    Calendar currentTime = Calendar.getInstance();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面的按钮
        setTime = (Button) findViewById(R.id.setTime);
        // 获取 AlarmManager 对象
        aManager = (AlarmManager) getSystemService(
            Service.ALARM_SERVICE);
        // 为“设置闹铃”按钮绑定监听器。
        setTime.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
```

```
                Calendar currentTime = Calendar.getInstance();
                // 创建一个TimePickerDialog实例，并把它显示出来。
                new TimePickerDialog(AlarmTest.this, 0, // 绑定监听器
                        new TimePickerDialog.OnTimeSetListener()
                        {
                            @Override
                            public void onTimeSet(TimePicker tp,
                                int hourOfDay, int minute)
                            {
                                // 指定启动AlarmActivity组件
                                Intent intent = new Intent(AlarmTest.this,
                                    AlarmActivity.class);
                                // 创建PendingIntent对象
                                PendingIntent pi = PendingIntent.getActivity(
                                    AlarmTest.this, 0, intent, 0);
                                Calendar c = Calendar.getInstance();
                                c.setTimeInMillis(System.currentTimeMillis());
                                // 根据用户选择时间来设置Calendar对象
                                c.set(Calendar.HOUR, hourOfDay);
                                c.set(Calendar.MINUTE, minute);
                                // 设置AlarmManager将在Calendar对应的时间启动指定组件
                                aManager.set(AlarmManager.RTC_WAKEUP,
                                    c.getTimeInMillis(), pi);
                                // 显示闹铃设置成功的提示信息
                                Toast.makeText(AlarmTest.this, "闹铃设置成功啦"
                                    , Toast.LENGTH_SHORT).show();
                            }
                        }, currentTime.get(Calendar.HOUR_OF_DAY), currentTime
                            .get(Calendar.MINUTE), false).show();
            }
        });
    }
}
```

上面的程序中粗体字代码控制AlarmManager将会在Calendar对应的时间启动pi对应的Activity组件，而且程序设置了AlarmManager.RTC_WAKEUP选项，这意味着即使系统处于关机状态，到了系统预设时间，AlarmManager也会控制系统去执行pi对应的Activity组件。

上面的程序中AlarmManager需要启动的Activity为AlarmActivity，它是一个非常简单的Activity，甚至不需要程序程序界面，当该Activity加载时打开一个对话框提示闹钟时间到，并播放一段“激昂”的音乐提醒用户。AlarmActivity程序代码如下。

程序清单：codes\10\10.7\AlarmTest\src\org\crazyit\manager\AlarmActivity.java

```
public class AlarmActivity extends Activity
{
    MediaPlayer alarmMusic;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 加载指定音乐，并为之创建MediaPlayer对象
        alarmMusic = MediaPlayer.create(this, R.raw.alarm);
        alarmMusic.setLooping(true);
        // 播放音乐
        alarmMusic.start();
        // 创建一个对话框
        new AlertDialog.Builder(AlarmActivity.this).setTitle("闹钟")
            .setMessage("闹钟响了,Go! Go! Go! ")
            .setPositiveButton("确定", new OnClickListener()
            {
```

```
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                    // 停止音乐
                    alarmMusic.stop();
                    // 结束该 Activity
                    AlarmActivity.this.finish();
                }
            }).show();
        }
    }
```

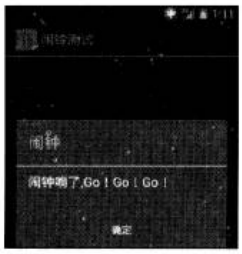


图 10.15　闹钟应用

上面的 Activity 的作用只是通过一个对话框、一段音乐来提醒用户：闹钟时间到了。

不要忘记在 AndroidManifest.xml 文件中配置 AlarmActivity，如果用户忘记配置该 Activity，系统甚至不会提示用户：该 Activity 不存在！只是到了系统预设时间之后，程序将什么也不干。

运行该程序，简单地把闹钟时间设为下一分钟，即使我们退出该程序，接下来过一分钟之后将可以看到如图 10.15 所示的闹钟提示。

实例：定时更换壁纸

本例程序将会通过 AlarmManager 来周期性地调用某个 Service，从而让系统实现定时更换壁纸的功能。

更换壁纸的 API 为 WallpaperManager，它提供了 clear()方法来清除壁纸，还提供了如下方法来设置系统的壁纸。

- setBitmap(Bitmap bitmap)：将壁纸设置 bitmap 所代表的位图。
- setResource(int resid)：将壁纸设为 resid 资源所代表的图片。
- setStream(InputStream data)：将壁纸设置 data 数据所代表的图片。

该程序的界面中只有两个按钮，一个按钮用于启动定时更换壁纸，另一个按钮用于关闭定时更换壁纸。该程序的代码如下。

程序清单：codes\10\10.7\AlarmChangeWallpaper\src\org\crazyit\manager\AlarmChange- Wallpaper.java

```
public class AlarmChangeWallpaper extends Activity
{
    // 定义 AlarmManager 对象
    AlarmManager aManager;
    Button start, stop;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        aManager = (AlarmManager) getSystemService(
            Service.ALARM_SERVICE);
        // 指定启动 ChangeService 组件
        Intent intent = new Intent(AlarmChangeWallpaper.this,
            ChangeService.class);
        // 创建 PendingIntent 对象
        final PendingIntent pi = PendingIntent.getService(
```

```
                AlarmChangeWallpaper.this, 0, intent, 0);
        start.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                // 设置每隔5秒执行pi代表的组件一次
                aManager.setRepeating(AlarmManager.RTC_WAKEUP
                    , 0, 5000, pi);
                start.setEnabled(false);
                stop.setEnabled(true);
                Toast.makeText(AlarmChangeWallpaper.this
                    , "壁纸定时更换启动成功啦",
                    Toast.LENGTH_SHORT).show();
            }
        });
        stop.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                start.setEnabled(true);
                stop.setEnabled(false);
                // 取消对pi的调度
                aManager.cancel(pi);
            }
        });
    }
}
```

上面的程序中粗体字代码指定程序每隔5秒执行一次pi所代表组件。程序中pi代表了ChangeService组件，因此还需要为该应用程序提供一个ChangeService组件，该组件的代码如下。

程序清单：codes\10\10.7\AlarmChangeWallpaper\src\org\crazyit\manager\ChangeService.java

```
public class ChangeService extends Service
{
    // 定义定时更换的壁纸资源
    int[] wallpapers = new int[]{
        R.drawable.shuangta,
        R.drawable.lijiang,
        R.drawable.qiao,
        R.drawable.shui
    };
    // 定义系统的壁纸管理服务
    WallpaperManager wManager;
    // 定义当前所显示的壁纸
    int current = 0;
    @Override
    public int onStartCommand(Intent intent, int flags, int startId)
    {
        // 如果到了最后一张，系统重新开始
        if(current >= 4)
            current = 0;
        try
        {
            // 改变壁纸
            wManager.setResource(wallpapers[current++]);
        }
        catch (Exception e)
```

```
            {
                e.printStackTrace();
            }
            return START_STICKY;
        }
        @Override
        public void onCreate()
        {
            super.onCreate();
            // 初始化 WallpaperManager
            wManager = WallpaperManager.getInstance(this);
        }
        @Override
        public IBinder onBind(Intent intent)
        {
            return null;
        }
    }
```

上面的程序重写了 Service 的 onStartCommand()方法，这意味着程序每次启动该 Service 都会执行 onStartCommand()方法中的代码，而 onStartCommand()方法中的粗体字代码负责更换系统的壁纸。

为了允许该程序改变壁纸，还需要在 AndroidManifest.xml 文件中为该程序授予相应的权限，也就是在 AndroidManifest.xml 文件中增加如下代码：

```
<!-- 授予用户修改壁纸的权限 -->
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
```

运行该程序，并单击程序界面上的“启动定时更换”按钮，接着退出该程序，返回到程序桌面，将可以看到系统桌面每 5 秒更换一次，如图 10.16 所示。



图 10.16　定时更换壁纸

当然这个程序还有值得改进的地方，这个程序所更换的壁纸只是程序预设的几张图片，如果把这个程序与前面介绍的 SD 卡文件浏览器结合起来，让用户可以添加 SD 卡中的图片作为可供更换的壁纸图片，那么这个程序就算比较完善了。

10.8　接收广播消息

Android 系统的四大组件还有一种 BroadcastReceiver，这种组件本质上就是一种全局的监听器，用于监听系统全局的广播消息。由于 BroadcastReceiver 是一种全局的监听器，因此它可以非常方便地实现系统中不同组件之间的通信。例如我们希望客户端程序与 startService()方法启动的 Service 之间通信，就可以借助于 BroadcastReceiver 来实现。

➤➤10.8.1　BroadcastReceiver 简介

BroadcastReceiver 用于接收程序（包括用户开发的程序和系统内建的程序）所发出的 Broadcast Intent，与应用程序启动 Activity、Service 相同的是，程序启动 BroadcastReceiver 也只需要两步。

① 创建需要启动的 BroadcastReceiver 的 Intent。

② 调用 Context 的 sendBroadcast()或 sendOrderedBroadcast()方法来启动指定的 BroadcastReceiver。

当应用程序发出一个 Broadcast Intent 之后，所有匹配该 Intent 的 BroadcastReceiver 都有可能被启动。

与 Activity、Service 具有完整的生命周期不同，BroadcastReceiver 本质上只是一个系统级的监听器——它专门负责监听各程序所发出的 Broadcast。



提示：

前面介绍的各种 OnXxxListener 只是程序级别的监听器，这些监听器运行在指定程序所在进程中，当程序退出时，OnXxxListener 监听器也就随之关闭了。但 BroadcastReceiver 属于系统级的监听器，它拥有自己的进程，只要存在与之匹配的 Intent 被广播出来，BroadcastReceiver 总会被激发。

由于 BroadcastReceiver 本质上属于一个监听器，因此实现 BroadcastReceiver 的方法也十分简单，只要重写 BroadcastReceiver 的 onReceive(Context context, Intent intent)方法即可。

一旦实现了 BroadcastReceiver，接下来就应该指定该 BroadcastReceiver 能匹配的 Intent，此时有两种方式。

- 使用代码进行指定，调用 BroadcastReceiver 的 Context 的 registerReceiver (BroadcastReceiver receiver, IntentFilter filter)方法指定。例如如下代码：

```
IntentFilter filter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
IncomingSMSReceiver receiver = new IncomingSMSReceiver();
registerReceiver(receiver, filter);
```

- 在 AndroidManifest.xml 文件中配置。例如如下代码：

```
<receiver android:name=".IncomingSMSReceiver">
    <intent-filter>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
    </intent-filter>
</receiver>
```

每次系统 Broadcast 事件发生后，系统就会创建对应的 BroadcastReceiver 的实例，并自动触发它的 onReceive() 方法，onReceive()方法执行完后，BroadcastReceiver 的实例就会被销毁。



提示：

与 Activity 组件不同的是，当系统通过 Intent 启动指定了 Activity 组件时，如果系统没有找到合适的 Activity 组件，会导致程序异常中止；但系统通过 Intent 激发 BroadcastReceiver 时，如果找不到合适的 BroadcastReceiver 组件，应用不会有任何问题。

如果 BroadcastReceiver 的 onReceive()方法不能在 10 秒内执行完成，Android 会认为该程序无响应。所以不要在 BroadcastReceiver 的 onReceive()方法里执行一些耗时的操作，否则会弹出 ANR（Application No Response）的对话框。

如果确实需要根据 Broadcast 来完成一项比较耗时的操作，则可以考虑通过 Intent 启动一

个 Service 来完成该操作。不应考虑使用新线程去完成耗时的操作，因为 BroadcastReceiver 本身的生命周期很短，可能出现的情况是子线程可能还没有结束，BroadcastReceiver 就已经退出了。

如果 BroadcastReceiver 所在的进程结束了，虽然该进程内还有用户启动的新线程，但由于该进程内不包含任何活动组件，因此系统可能在内存紧张时优先结束该进程。这样就可能导致 BroadcastReceiver 启动的子线程不能执行完成。

10.8.2 发送广播

在程序中发送广播十分简单，只要调用 Context 的 sendBroadcast(Intent intent)方法即可，这条广播将会启动 intent 参数所对应的 BroadcastReceiver。

下面一个简单的程序示范了如何发送 Broadcast、使用 BroadcastReceiver 接收广播，该程序的 Activity 界面中包含一个按钮，当用户单击该按钮时程序会向外发送一条广播。该程序的代码如下。

程序清单：codes\10\10.8\Broadcast\src\org\crazyit\broadcast\BroadcastMain.java

```
public class BroadcastMain extends Activity
{
    Button send;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的按钮
        send = (Button) findViewById(R.id.send);
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                // 设置 Intent 的 Action 属性
                intent.setAction("org.crazyit.action.CRAZY_BROADCAST");
                intent.putExtra("msg", "简单的消息");
                // 发送广播
                sendBroadcast(intent);
            }
        });
    }
}
```

上面的程序中粗体字代码用于创建一个 Intent 对象，并使用该 Intent 对象对外发送一条广播，该程序所使用的 BroadcastReceiver 代码如下。

程序清单：codes\10\10.8\Broadcast\src\org\crazyit\broadcast\MyReceiver.java

```
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context,
            "接收到的 Intent 的 Action 为：" + intent.getAction()
```

```
                + "\n 消息内容是：" + intent.getStringExtra("msg")
                , Toast.LENGTH_LONG).show();
        }
    }
```

正如上面的程序中看到的，当符合该 MyReceiver 的广播出现时，该 MyReceiver 的 onReceive()方法将会被触发，从而在该方法中显示广播所携带的消息。

上面发送广播的程序中指定发送广播时所用的 Intent 的 Action 为 org.crazyit.action.CRAZY_BROADCAST，这就需要配置上面的 BroadcastReceiver 应监听 Action 为该字符串的 Intent，在 AndroidManifest.xml 文件中增加如下配置即可：

```
<receiver android:name=".MyReceiver">
    <intent-filter>
        <!-- 指定该 BroadcastReceiver 所响应的 Intent 的 Action -->
        <action android:name="org.crazyit.action.CRAZY_BROADCAST" />
    </intent-filter>
</receiver>
```

运行该程序，并单击程序中的“发送广播”按钮，将看到程序有如图 10.17 所示的提示。

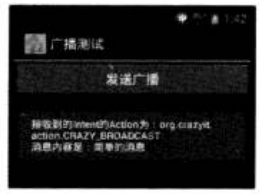


图 10.17 响应广播

10.8.3 有序广播

Broadcast 被分为如下两种。

- Normal Broadcast（普通广播）：Normal Broadcast 是完全异步的，可以在同一时刻（逻辑上）被所有接收者接收到，消息传递的效率比较高。但缺点是接收者不能将处理结果传递给下一个接收者，并且无法终止 Broadcast Intent 的传播。
- Ordered Broadcast（有序广播）：Ordered Broadcast 的接收者将按预先声明的优先级依次接收 Broadcast。如：A 的级别高于 B、B 的级别高于 C，那么，Broadcast 先传给 A，再传给 B，最后传给 C 。优先级别声明在<intent-filter.../>元素的 android:priority 属性中，数越大优先级别越高，取值范围为-1000～1000，优先级别也可以调用 IntentFilter 对象的 setPriority()进行设置。Ordered Broadcast 接收者可以终止 Broadcast Intent 的传播，Broadcast Intent 的传播一旦终止，后面的接收者就无法接收到 Broadcast。另外，Ordered Broadcast 的接收者可以将数据传递给下一个接收者，如：A 得到 Broadcast 后，可以往它的结果对象中存入数据，当 Broadcast 传给 B 时，B 可以从 A 的结果对象中得到 A 存入的数据。

Context 提供的如下两个方法用于发送广播。

- sendBroadcast()：发送 Normal Broadcast。
- sendOrderedBroadcast()：发送 Ordered Broadcast。

对于 Ordered Broadcast 而言，系统会根据接收者声明的优先级别按顺序逐个执行接收者，优先接收到 Broadcast 的接收者可以终止 Broadcast，调用 BroadcastReceiver 的 abortBroadcast()方法即可终止 Broadcast。如果 Broadcast 被前面的接收者终止，后面的接收者就再也无法获取到 Broadcast。

不仅如此，对于 Ordered Broadcast 而言，优先接收到 Broadcast 的接收者可以通过 setResultExtras(Bundle)方法将处理结果存入 Broadcast 中，然后传给下一个接收者，下一个接收者通过代码：Bundle bundle = getResultExtras(true)可以获取上一个接收者存入的数据。



提示：
系统收到短信，发出的 Broadcast 属于 Ordered Broadcast。如果想阻止用户收到短信，可以通过设置优先级，让自定义的 BroadcastReceiver 先获取到 Broadcast，然后终止 Broadcast。

接下来介绍一个发送有序广播的示例，该程序的 Activity 界面上只有一个普通按钮，该按钮用于发送一条有序广播。该程序代码如下。

程序清单：codes\10\10.8\SortedBroadcast\src\org\crazyit\broadcast\SortedBroadcast.java

```
public class SortedBroadcast extends Activity
{
    Button send;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序中的 send 按钮
        send = (Button) findViewById(R.id.send);
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 创建 Intent 对象
                Intent intent = new Intent();
                intent.setAction("org.crazyit.action.CRAZY_BROADCAST");
                intent.putExtra("msg", "简单的消息");
                // 发送有序广播
                sendOrderedBroadcast(intent, null);
            }
        });
    }
}
```

上面的程序中粗体字代码指定了 Intent 的 Action 属性，再调用 sendOrderedBroadcast()方法来发送有序广播。对于有序广播而言，它会按优先级依次触发每个 BroadcastReceiver 的 onReceive()方法。

下面的程序先定义第一个 BroadcastReceiver。

程序清单：codes\10\10.8\SortedBroadcast\src\org\crazyit\broadcast\MyReceiver.java

```
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context,    "接收到的 Intent 的 Action 为："
            + intent.getAction() + "\n 消息内容是："
            + intent.getStringExtra("msg")
            , Toast.LENGTH_LONG).show();
        // 创建一个 Bundle 对象，并存入数据
        Bundle bundle = new Bundle();
        bundle.putString("first", "第一个 BroadcastReceiver 存入的消息");
        // 将 bundle 放入结果中
        setResultExtras(bundle);
        // 取消 Broadcast 的继续传播
        // abortBroadcast(); //①
```

```
        }
    }
```

上面的 BroadcastReceiver 不仅处理了它所接收到的消息，而且向处理结果中存入了 key 为 first 的消息，这个消息将可以被第二个 BroadcastReceiver 解析出来。

上面的程序中①号粗体字代码用于取消广播，如果保持这条代码生效，那么优先级比 MyReceiver 低的 BroadcastReceiver 都将不会被触发。

在 AndroidManifest.xml 文件中部署该 BroadcastReceiver，并指定其优先级为 20，配置片段如下：

```
<receiver android:name=".MyReceiver">
    <intent-filter android:priority="20">
        <action android:name="org.crazyit.action.CRAZY_BROADCAST" />
    </intent-filter>
</receiver>
```

接下来再为程序提供第二个 BroadcastReceiver，这个 BroadcastReceiver 将会解析前一个 BroadcastReceiver 所存入的 key 为 first 的消息。该 BroadcastReceiver 的代码如下。

程序清单：codes\10\10.8\SortedBroadcast\src\org\crazyit\broadcast\MyReceiver2.java

```
public class MyReceiver2 extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = getResultExtras(true);
        // 解析前一个 BroadcastReceiver 所存入的 key 为 first 的消息
        String first = bundle.getString("first");
        Toast.makeText(context, "第一个 Broadcast 存入的消息为："
            + first, Toast.LENGTH_LONG).show();
    }
}
```

上面的程序中粗体字代码用于解析前一个 BroadcastReceiver 存入结果中 key 为 first 的消息，在 AndroidManifest.xml 文件中配置该 BroadcastReceiver，并指定其优先级为 0。配置片段如下：

```
<receiver android:name=".MyReceiver2">
    <intent-filter android:priority="0">
        <action android:name="org.crazyit.action.CRAZY_BROADCAST" />
    </intent-filter>
</receiver>
```

根据上面的配置可以看出，该程序中包含两个 Broadcast Receiver，其中 MyReceiver 的优先级更高，MyReceiver2 的优先级略低。如果将程序中①号粗体字代码注释掉，那么程序中 MyReceiver2 将可以被触发，并解析得到 MyReceiver 存入结果中的 key 为 first 的消息，因此看到如图 10.18 所示的输出。



图 10.18 获取前一个 BroadcastReceiver 存入结果中的消息

如果不注释程序中①号粗体字代码，这行代码将会阻止消息广播，这样消息将传不到 MyReceiver2 了。

实例：基于 Service 的音乐播放器

前面已经提到 BroadcastReceiver 是一种全局监听器，因此 BroadcastReceiver 也就提供了

让不同组件之间进行通信的新思路，比如程序有一个 Activity、一个 Service，而且该 Service 是通过 startService()方法启动起来的，正常情况下，这个 Activity 与通过 startService()方法启动的 Service 之间无法通信，但借助于 BroadcastReceiver 的帮助，程序就可以实现两者之间的通信了。

下面的程序开发了一个基于 Service 组件的音乐盒，程序的音乐将会由后台运行的 Service 组件负责播放，当后台的播放状态发生改变时，程序将会通过发送广播通知前台 Activity 更新界面；当用户单击前台 Activity 的界面按钮时，系统将通过发送广播通知后台 Service 来改变播放状态。

前台 Activity 的界面很简单，它只有两个按钮，分别用于控制播放/暂停、停止，另外还有两个文本框，用于显示正在播放的歌曲名、歌手名。前台 Activity 的代码如下。

程序清单：codes\10\10.8\MusicBox\src\org\crazyit\broadcast\MusicBox.java

```
public class MusicBox extends Activity implements OnClickListener
{
    // 获取界面中显示歌曲标题、作者文本框
    TextView title, author;
    // 播放/暂停、停止按钮
    ImageButton play, stop;
    ActivityReceiver activityReceiver;
    public static final String CTL_ACTION =
        "org.crazyit.action.CTL_ACTION";
    public static final String UPDATE_ACTION =
        "org.crazyit.action.UPDATE_ACTION";
    // 定义音乐的播放状态，0x11 代表没有播放；0x12 代表正在播放；0x13 代表暂停
    int status = 0x11;
    String[] titleStrs = new String[] { "心愿", "约定", "美丽新世界" };
    String[] authorStrs = new String[] { "未知艺术家", "周蕙", "伍佰" };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面界面中的两个按钮
        play = (ImageButton) this.findViewById(R.id.play);
        stop = (ImageButton) this.findViewById(R.id.stop);
        title = (TextView) findViewById(R.id.title);
        author = (TextView) findViewById(R.id.author);
        // 为两个按钮的单击事件添加监听器
        play.setOnClickListener(this);
        stop.setOnClickListener(this);
        activityReceiver = new ActivityReceiver();
        // 创建 IntentFilter
        IntentFilter filter = new IntentFilter();
        // 指定 BroadcastReceiver 监听的 Action
        filter.addAction(UPDATE_ACTION);
        // 注册 BroadcastReceiver
        registerReceiver(activityReceiver, filter);
        Intent intent = new Intent(this, MusicService.class);
        // 启动后台 Service
        startService(intent);
    }
    // 自定义的 BroadcastReceiver，负责监听从 Service 传回来的广播
    public class ActivityReceiver extends BroadcastReceiver
    {
        @Override
        public void onReceive(Context context, Intent intent)
```

```
            {
                // 获取 Intent 中的 update 消息，update 代表播放状态
                int update = intent.getIntExtra("update", -1);
                // 获取 Intent 中的 current 消息，current 代表当前正在播放的歌曲
                int current = intent.getIntExtra("current", -1);
                if (current >= 0)
                {
                    title.setText(titleStrs[current]);
                    author.setText(authorStrs[current]);
                }
                switch (update)
                {
                    case 0x11:
                        play.setImageResource(R.drawable.play);
                        status = 0x11;
                        break;
                    // 控制系统进入播放状态
                    case 0x12:
                        // 播放状态下设置使用暂停图标
                        play.setImageResource(R.drawable.pause);
                        // 设置当前状态
                        status = 0x12;
                        break;
                    // 控制系统进入暂停状态
                    case 0x13:
                        // 暂停状态下设置使用播放图标
                        play.setImageResource(R.drawable.play);
                        // 设置当前状态
                        status = 0x13;
                        break;
                }
            }
        }
        @Override
        public void onClick(View source)
        {
            // 创建 Intent
            Intent intent = new Intent("org.crazyit.action.CTL_ACTION");
            switch (source.getId())
            {
                // 按下播放/暂停按钮
                case R.id.play:
                    intent.putExtra("control", 1);
                    break;
                // 按下停止按钮
                case R.id.stop:
                    intent.putExtra("control", 2);
                    break;
            }
            // 发送广播，将被 Service 组件中的 BroadcastReceiver 接收到
            sendBroadcast(intent);
        }
}
```

上面程序的关键代码就是两段粗体字代码：

- 第一段粗体字代码用于响应后台 Service 所发出的广播，该程序将会根据广播 Intent 里的消息来改变播放状态，并更新程序界面中按钮的图标：当正在播放时，显示暂停图标；当正在暂停时，显示播放图表。并根据传回来的 current 数据来更新 title、author 两个文本框所显示的文本——显示当前正在播放的歌曲的歌名和歌手。

- 第二段粗体字代码则根据用户单击的按钮发送广播，发送广播时会把所按下的按钮的标识发送出来。发送的广播将激发后台 Service 的 BroadcastReceiver，该 BroadcastReceiver 将会根据广播消息来改变播放状态。

与之对应的是，该程序的后台 Service 也一样，它会在播放状态发生改变时对外发送广播（广播将会激发前台 Activity 的 BroadcastReceiver）；它也会采用 BroadcastReceiver 监听来自前台 Activity 所发出的广播。后台 Service 的代码如下。

程序清单：codes\10\10.8\MusicBox\src\org\crazyit\broadcast\MusicService.java

```
public class MusicService extends Service
{
    MyReceiver serviceReceiver;
    AssetManager am;
    String[] musics = new String[] { "wish.mp3", "promise.mp3",
        "beautiful.mp3" };
    MediaPlayer mPlayer;
    // 当前的状态，0x11 代表没有播放；0x12 代表正在播放；0x13 代表暂停
    int status = 0x11;
    // 记录当前正在播放的音乐
    int current = 0;
    @Override
    public IBinder onBind(Intent intent)
    {
        return null;
    }
    @Override
    public void onCreate()
    {
        am = getAssets();
        // 创建 BroadcastReceiver
        serviceReceiver = new MyReceiver();
        // 创建 IntentFilter
        IntentFilter filter = new IntentFilter();
        filter.addAction(MusicBox.CTL_ACTION);
        registerReceiver(serviceReceiver, filter);
        // 创建 MediaPlayer
        mPlayer = new MediaPlayer();
        // 为 MediaPlayer 播放完成事件绑定监听器
        mPlayer.setOnCompletionListener(new OnCompletionListener() //①
        {
            @Override
            public void onCompletion(MediaPlayer mp)
            {
                current++;
                if (current >= 3)
                {
                    current = 0;
                }
                //发送广播通知 Activity 更改文本框
                Intent sendIntent = new Intent(MusicBox.UPDATE_ACTION);
                sendIntent.putExtra("current", current);
                // 发送广播，将被 Activity 组件中的 BroadcastReceiver 接收到
                sendBroadcast(sendIntent);
                // 准备并播放音乐
                prepareAndPlay(musics[current]);
            }
        });
        super.onCreate();
    }
```

```
public class MyReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(final Context context, Intent intent)
    {
        int control = intent.getIntExtra("control", -1);
        switch (control)
        {
            // 播放或暂停
            case 1:
                // 原来处于没有播放状态
                if (status == 0x11)
                {
                    // 准备并播放音乐
                    prepareAndPlay(musics[current]);
                    status = 0x12;
                }
                // 原来处于播放状态
                else if (status == 0x12)
                {
                    // 暂停
                    mPlayer.pause();
                    // 改变为暂停状态
                    status = 0x13;
                }
                // 原来处于暂停状态
                else if (status == 0x13)
                {
                    // 播放
                    mPlayer.start();
                    // 改变状态
                    status = 0x12;
                }
                break;
            // 停止声音
            case 2:
                // 如果原来正在播放或暂停
                if (status == 0x12 || status == 0x13)
                {
                    // 停止播放
                    mPlayer.stop();
                    status = 0x11;
                }
        }
        // 广播通知Activity更改图标、文本框
        Intent sendIntent = new Intent(MusicBox.UPDATE_ACTION);
        sendIntent.putExtra("update", status);
        sendIntent.putExtra("current", current);
        // 发送广播，将被Activity组件中的BroadcastReceiver接收到
        sendBroadcast(sendIntent);
    }
}
private void prepareAndPlay(String music)
{
    try
    {
        // 打开指定音乐文件
        AssetFileDescriptor afd = am.openFd(music);
        mPlayer.reset();
        // 使用MediaPlayer加载指定的声音文件。
        mPlayer.setDataSource(afd.getFileDescriptor(),
```

```
                    afd.getStartOffset(), afd.getLength());
                // 准备声音
                mPlayer.prepare();
                // 播放
                mPlayer.start();
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
    }
```

上面的程序中粗体字代码用于接收来自前台 Activity 所发出的广播，并根据广播的消息内容改变 Service 的播放状态，当播放状态改变时，该 Service 对外发送一条广播，广播消息将会被前台 Activity 接收，前台 Activity 将会根据广播消息去更新程序界面。

除此之外，为了让该音乐盒能按顺序依次播放每首歌曲，程序为 MediaPlayer 增加了 OnCompletionListener 监听器，当 MediaPlayer 播放完成后将让它自动播放下一首歌曲。如程序中①号粗体字代码所示。



图 10.19　音乐播放器

运行该程序，将看到如图 10.19 示的界面。

由于该程序采用了后台 Service 来播放音乐，因此即使用户退出该程序，但后台依然会播放音乐，这就是该程序的独特之处。如果用户希望停止播放，只要单击图 10.19 示界面中的“停止”按钮，前台 Activity 将会发送广播通知后台 Service 停止播放。

该程序用到了两个 BroadcastReceiver，但已经在程序中注册两个 BroadcastReceiver 所监听的 IntentFilter，因此无须在 AndroidManifest.xml 文件中注册这两个 BroadcastReceiver，只要注册该程序所用的前台 Activity、后台 Service 即可。

10.9　接收系统广播消息

除了接收用户发送的广播之外，BroadcastReceiver 还有一个重要的用途：接收系统广播。如果应用需要在系统特定时刻执行某些操作，就可以通过监听系统广播来实现。Android 的大量系统事件都会对外发送标准广播。下面是 Android 常见的广播 Action 常量（具体请参考 Android API 文档中关于 Intent 的说明）。

- ACTION_TIME_CHANGED：系统时间被改变。
- ACTION_DATE_CHANGED：系统日期被改变。
- ACTION_TIMEZONE_CHANGED：系统时区被改变。
- ACTION_BOOT_COMPLETED：系统启动完成。
- ACTION_PACKAGE_ADDED：系统添加包。
- ACTION_PACKAGE_CHANGED：系统的包改变。
- ACTION_PACKAGE_REMOVED：系统的包被删除。
- ACTION_PACKAGE_RESTARTED：系统的包被重启。
- ACTION_PACKAGE_DATA_CLEARED：系统的包数据被清空。
- ACTION_BATTERY_CHANGED：电池电量改变。
- ACTION_BATTERY_LOW：电池电量低。
- ACTION_POWER_CONNECTED：系统连接电源。

- ACTION_POWER_DISCONNECTED：系统与电源断开。
- ACTION_SHUTDOWN：系统被关闭。

通过使用 BroadcastReceiver 来监听特殊的广播，即可让应用随系统执行特定的操作。

实例：开机自动运行的 Service

前面已经介绍了多个程序，需要使用开机自动运行的 Service，例如监听用户来电、监听用户短信、拦截黑名单电话……为了让 Service 随应系统启动自动运行，可以让 BroadcastReceiver 监听 Action 为 ACTION_BOOT_COMPLETED 常量的 Intent，然后在 BroadcastReceiver 中启动特定 Service 即可。

该程序所用的 BroadcastReceiver 代码如下。

程序清单：codes\10\10.9\LaunchService\src\org\crazyit\broadcast\LaunchReceiver.java

```
public class LaunchReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Intent tIntent = new Intent(context
            , LaunchService.class);
        // 启动指定 Service
        context.startService(tIntent);
    }
}
```

该 LaunchReceiver 的代码十分简单，只要在 onReceive()方法中启动指定 Service 即可，关键是要让 LaunchReceiver 监听系统开机发出的广播，因此需要在 AndroidManifest.xml 文件中采用如下代码配置该 BroadcastReceiver：

```
<!-- 定义一个 BroadcastReceiver,监听系统开机广播 -->
<receiver android:name=".LaunchReceiver">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

除此之外，为了让程序能访问系统开机事件，还需要为应用程序增加如下权限：

```
<!-- 授予应用程序访问系统开机事件的权限 -->
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

至于程序中用到的 LaunchService，则可以是用户开发的任意 Service，既可是监听用户来电的 Service，也可以是监听用户短信、拦截黑名单电话等的 Service，此处不再给出该 Service 的代码。

实例：短信提醒

前面已经提到，当系统收到短信时，系统会对外发送一个有序广播，该广播的 Intent 的 Action 为 android.provider.Telephony.SMS_RECEIVED。因此只要在程序中开发一个对应的 BroadcastReceiver 即可监听到系统收到的短信。

该程序对应的 BroadcastReceiver 代码如下。

程序清单：codes\10\10.9\MonitorSms\src\org\crazyit\broadcast\SmsReceiver.java

```
public class SmsReceiver extends BroadcastReceiver
```

```
{
    // 当接收到短信时被触发
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 如果是接收到短信
        if (intent.getAction().equals(
            "android.provider.Telephony.SMS_RECEIVED"))
        {
            // 取消广播（这行代码将会让系统收不到短信）
            abortBroadcast();  //①
            StringBuilder sb = new StringBuilder();
            // 接收由 SMS 传过来的数据
            Bundle bundle = intent.getExtras();
            // 判断是否有数据
            if (bundle != null)
            {
                // 通过 pdus 可以获得接收到的所有短信消息
                Object[] pdus = (Object[]) bundle.get("pdus");
                // 构建短信对象 array,并依据收到的对象长度来创建 array 的大小
                SmsMessage[] messages = new SmsMessage[pdus.length];
                for (int i = 0; i < pdus.length; i++)
                {
                    messages[i] = SmsMessage
                        .createFromPdu((byte[]) pdus[i]);
                }
                // 将发送来的短信合并自定义信息于 StringBuilder 当中
                for (SmsMessage message : messages)
                {
                    sb.append("短信来源:");
                    // 获得接收短信的电话号码
                    sb.append(message.getDisplayOriginatingAddress());
                    sb.append("\n------短信内容------\n");
                    // 获得短信的内容
                    sb.append(message.getDisplayMessageBody());
                }
            }
            Toast.makeText(context, sb.toString()
                , Toast.LENGTH_LONG).show();
        }
    }
}
```

上面的程序重写了 BroadcastReceiver 的 onReceive()方法，并在该方法中获取所收到的短信的内容，然后使用 Toast 显示短信内容。

为了让该程序在系统的短信接收程序之前被启动，我们将该 BroadcastReceiver 的优先级设得高一些，在 AndroidManifest.xml 文件中通过如下代码来配置该 BroadcastReceiver：

```
<receiver android:name="SmsReceiver">
    <intent-filter android:priority="800">
        <action android:name="android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

上面的配置片段中粗体字代码指定了该 BroadcastReceiver 的优先级为 800，这样它就可以在系统短信接收程序之前被触发。为了让该程序拥有读取短信的权限，还需要在 AndroidManifest.xml 文件中为该程序进行授权，授权的配置代码如下：

```
<!-- 授予程序接收短信的权限 -->
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
```

由于该 BroadcastReceiver 将会位于系统短信接收程序之前被启动，如果保持该程序中①号代码生效，那么该程序接收到短信广播之后，广播将不会被传播到系统的短信接收程序——也就是系统本身将不会收到短信。

运行该程序，然后再启动另一个模拟器向该程序所在模拟器发送短信，将可看到如图 10.20 所示的界面。



图 10.20 短信提醒

当然，该程序也可改变成所谓“短信窃听”程序，只要让该程序接收到短信之后并不将该短信显示出来，而是将短信存入系统中，或者再以短信的方式发送到指定手机，这样就实现了“短信窃听”功能，不过这样做是“不道德”的。

实例：手机电量提示

当手机电量发生改变时，系统会对外发送 Intent 的 Action 为 ACTION_ BATTERY_ CHANGED 常量的广播；当手机电量过低时，系统会对外发送 Intent 的 Action 为 ACTION_ BATTERY_LOW 常量的广播。通过开发监听对应 Intent 的 BroadcastReceiver，即可让系统对手机电量进行提示。

下面的程序即可对手机电量过低进行提示。

程序清单：codes\10\10.9\MonitorBattery\src\org\crazyit\broadcast\BatteryReceiver.java

```
public class BatteryReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Bundle bundle = intent.getExtras();
        // 获取当前电量
        int current = bundle.getInt("level");
        // 获取总电量
        int total = bundle.getInt("scale");
        // 如果当前电量小于总电量的 15%
        if (current * 1.0 / total < 0.15)
        {
            Toast.makeText(context, "电量过低，请尽快充电！"
                , Toast.LENGTH_LONG).show();
        }
    }
}
```

上面的程序中粗体字代码即可通过接收到的广播消息获取系统总电量、当前电量，如果当前电量小于总电量的 15%，则程序会显示电量过低的提示信息。

10.10 本章小结

前面已经介绍了 Activity 和 ContentProvider 两个组件，再加上本章所介绍的 Service、BroadcastReceiver 两个组件，就是 Android 系统的四大组件了。学习 Service 需要重点掌握创建、配置 Service 组件，以及如何启动、停止 Service；不仅如此，如何开发 IntentService、远程 AIDL Service、调用远程 AIDL Service 也是需要重点掌握的内容。学习 BroadcastReceiver 需要掌握创建、配置 BroadcastReceiver 组件，还需要掌握在程序中发送 Broadcast 的方法。除此之外，本章还介绍了大量系统 Service 的功能和用法，包括 TelephonyManager、SmsManager、AudioManager、Vibrator、AlarmManager 等，需要读者熟练掌握并能熟练使用。

CHAPTER

11

第11章 多媒体应用开发

本章要点

- 音频和视频
- 使用 MediaPlayer 播放音频
- 使用 SoundPool 播放音频
- 使用 VideoView 播放视频
- 使用 MediaPlayer 与 SurfaceView 播放视频
- 使用 MediaRecorder 录制音频
- 控制摄像头拍照
- 控制摄像头录制视频短片

Android 应用面向的是普通个人用户，这些用户往往会更加关注用户体验，因此为 Android 应用增加动画、视频、音乐等多媒体功能十分必要。就目前的手机发展趋势来看，手机已经不再是单一的通信工具，已经发展成集照相机、音乐播放器、视频播放器、个人小型终端于一体的智能设备，因此为手机提供音频录制、播放，视频录制、播放的功能十分重要。

Android 提供了常见音频、视频的编码、解码机制，就像之前所用过的 MediaPlayer 类，Android 支持的音频格式有 MP3、WAV 和 3GP 等，支持的视频格式有 MP4 和 3GP 等。

借助于这些多媒体支持类，我们可以非常方便地在手机应用中播放音频、视频等，这些多媒体数据既可是来自于 Android 应用的资源文件，也可是来自于外部存储器上的文件，甚至可以是来自于网络的文件流。不仅如此，Android 也提供了对摄像头、麦克风的支持，因此也可以十分方便地从外部采集照片、视频、音频等多媒体信息。

11.1 音频和视频的播放

Android 提供了简单的 API 来播放音频、视频，下面将会详细介绍如何使用它们。

11.1.1 使用 MediaPlayer 播放音频

前面已经提及如何使用 MediaPlayer 播放音频的例子，使用 MediaPlayer 播放音频十分简单，当程序控制 MediaPlayer 对象装载音频完成之后，程序可以调用 MediaPlayer 的如下三个方法进行播放控制。

- start()：开始或恢复播放。
- stop()：停止播放。
- pause()：暂停播放。

为了让 MediaPlayer 来装载指定音频文件，MediaPlayer 提供了如下简单的静态方法。

- static MediaPlayer create(Context context, Uri uri)：从指定 Uri 来装载音频文件，并返回新创建的 MediaPlayer 对象。
- static MediaPlayer create(Context context, int resid)：从 resid 资源 ID 对应的资源文件中装载音频文件，并返回新创建的 MediaPlayer 对象。

上面这两个方法用起来非常方便，但这两个方法每次都会返回新创建的 MediaPlayer 对象，如果程序需要使用 MediaPlayer 循环播放多个音频文件，使用 MediaPlayer 的静态 create 方法就不太合适了，此时可通过 MediaPlayer 的 setDataSource()方法来装载指定的音频文件。MediaPlayer 提供了如下方法来指定装载相应的音频文件。

- setDataSource(String path)：指定装载 path 路径所代表的文件。
- setDataSource(FileDescriptor fd, long offset, long length)：指定装载 fd 所代表的文件中从 offset 开始、长度为 length 的文件内容。
- setDataSource(FileDescriptor fd)：指定装载 fd 所代表的文件。
- setDataSource(Context context, Uri uri)：指定装载 uri 所代表的文件。

执行上面所示的 setDataSource()方法之后，MediaPlayer 并未真正去装载那些音频文件，还需要调用 MediaPlayer 的 prepare()方法去准备音频，所谓“准备”，就是让 MediaPlayer 真正去装载音频文件。

因此使用已有的 MediaPlayer 对象装载“下一首”歌曲的代码模板为：

```
try
{
      mPlayer.reset();
      //装载下一首歌曲
      mPlayer.setDataSource("/mnt/sdcard/next.mp3);
      // 准备声音
      mPlayer.prepare();
      // 播放
      mPlayer.start();
}
catch (IOException e)
{
      e.printStackTrace();
}
```

除此之外，MediaPlayer 还提供了一些绑定事件监听器的方法，用于监听 MediaPlayer 播放过程中所发生的特定事件，绑定事件监听器的方法如下。

- setOnCompletionListener(MediaPlayer.OnCompletionListener listener)：为 MediaPlayer 的播放完成事件绑定事件监听器。
- setOnErrorListener(MediaPlayer.OnErrorListener listener)：为 MediaPlayer 的播放错误事件绑定事件监听器。
- setOnPreparedListener(MediaPlayer.OnPreparedListener listener)：当 MediaPlayer 调用 prepare()方法时触发该监听器。
- setOnSeekCompleteListener(MediaPlayer.OnSeekCompleteListener listener)：当 MediaPlayer 调用 seek()方法时触发该监听器。

因此可以在创建一个 MediaPlayer 对象之后，通过为该 MediaPlayer 绑定监听器来监听相应的事件，例如如下代码：

```
// 为 MediaPlayer 的播放完成事件绑定事件监听器
mPlayer.setOnErrorListener(new OnErrorListener ()
{
      @Override
      onError(MediaPlayer mp, int what, int extra)
      {
            //针对错误进行相应的处理
            ...
      }
});
// 为 MediaPlayer 的播放完成事件绑定事件监听器
mPlayer.setOnCompletionListener(new OnCompletionListener()
{
      @Override
      public void onCompletion(MediaPlayer mp)
      {
            current++;
            if (current >= 3)
            {
                  current = 0;
            }
            prepareAndPlay(musics[current]);
      }
});
```

下面简单归纳一下 MediaPlayer 播放不同来源的音频文件。

1. 播放应用的资源文件

播放应用的资源文件需要两步即可：

① 调用 MediaPlayer 的 create(Context context, int resid)方法加载指定资源文件。

② 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

例如如下代码：

```
MediaPlayer mPlayer = MediaPlayer.create(this, R.raw.song);
mPlayer.start();
```



提示： 音频资源文件一般放在 Android 应用的/res/raw 目录下。

2. 播放应用的原始资源文件

播放应用的资源文件按如下步骤执行。

① 调用 Context 的 getAssets()方法获取应用的 AssetManager。

② 调用 AssetManager 对象的 openFd(String name)方法打开指定的原生资源，该方法返回一个 AssetFileDescriptor 对象。

③ 调用 AssetFileDescriptor 的 getFileDescriptor()、getStartOffset()和 getLength()方法来获取音频文件的 FileDescriptor、开始位置、长度等。

④ 创建 MediaPlayer 对象（或利用已有的 MediaPlayer 对象），并调用 MediaPlayer 对象的 setDataSource(FileDescriptor fd, long offset, long length)方法来装载音频资源。

⑤ 调用 MediaPlayer 对象的 prepare()方法准备音频。

⑥ 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

注意： 虽然 MediaPlayer 提供了 setDataSource（FileDescriptor fd）方法来装载指定音频资源，但实际使用时这个方法似乎有问题：不管程序调用 openFd（String name）方法时指定打开哪个原始资源，MediaPlayer 将总是播放第一个原始的音频资源。



例如如下代码片段：

```
AssetManager am = getAssets();
// 打开指定音乐文件
AssetFileDescriptor afd = am.openFd(music);
MediaPlayer mPlayer = new MediaPlayer();
// 使用 MediaPlayer 加载指定的声音文件
mPlayer.setDataSource(afd.getFileDescriptor()
      , afd.getStartOffset()
      , afd.getLength());
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

3. 播放外部存储器上音频文件

播放外部存储器上音频文件按如下步骤执行。

① 创建 MediaPlayer 对象（或利用已有的 MediaPlayer 对象），并调用 MediaPlayer 对象的 setDateSource(String path)方法装载指定的音频文件。

② 调用 MediaPlayer 对象的 prepare()方法准备音频。

③ 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

例如如下代码：

```
MediaPlayer mPlayer = new MediaPlayer();
// 使用 MediaPlayer 加载指定的声音文件
mPlayer.setDataSource("/mnt/sdcard/mysong.mp3");
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

4．播放来自网络的音频文件

播放来自网络的音频文件有两种方式：① 直接使用 MediaPlayer 的静态 create(Context context, Uri uri)方法；② 调用 MediaPlayer 的 setDataSource(Context context, Uri uri)装载指定 Uri 对应的音频文件。

以第二种方式播放来自网络的音频文件的步骤如下。

① 根据网络上的音频文件所在的位置创建 Uri 对象。

② 创建 MediaPlayer 对象（或利用已有的 MediaPlayer 对象），并调用 MediaPlayer 对象的 setDateSource(Context context, Uri uri)方法装载 Uri 对应的音频文件。

③ 调用 MediaPlayer 对象的 prepare()方法准备音频。

④ 调用 MediaPlayer 的 start()、pause()、stop()等方法控制播放即可。

例如如下代码片段：

```
Uri uri = Uri.parse("http://www.crazyit.org/abc.mp3");
MediaPlayer mPlayer = new MediaPlayer();
// 使用 MediaPlayer 根据 Uri 来加载指定的声音文件
mPlayer.setDataSource(this, uri);
// 准备声音
mPlayer.prepare();
// 播放
mPlayer.start();
```

MediaPlayer 除了调用 prepare()方法来准备声音之外，还可以调用 prepareAsync()来准备声音，prepareAsync()与普通 prepare()方法的区别在于，prepareAsync()是异步的，它不会阻塞当前的 UI 线程。

归纳起来，MediaPlayer 有图 11.1 所示状态图。

由于前面已经提供了大量使用 MediaPlayer 播放声音的例子，故此处不再介绍使用 MediaPlayer 的简单方法。

➤➤11.1.2　音乐特效控制

Android 可以控制播放音乐时的均衡器、重低音、音场及显示音乐波形等，这些都是靠 AudioEffect 及其子类来完成的，它包含如下常用子类。

- AcousticEchoCanceler：取消回声控制器。
- AutomaticGainControl：自动增益控制器。

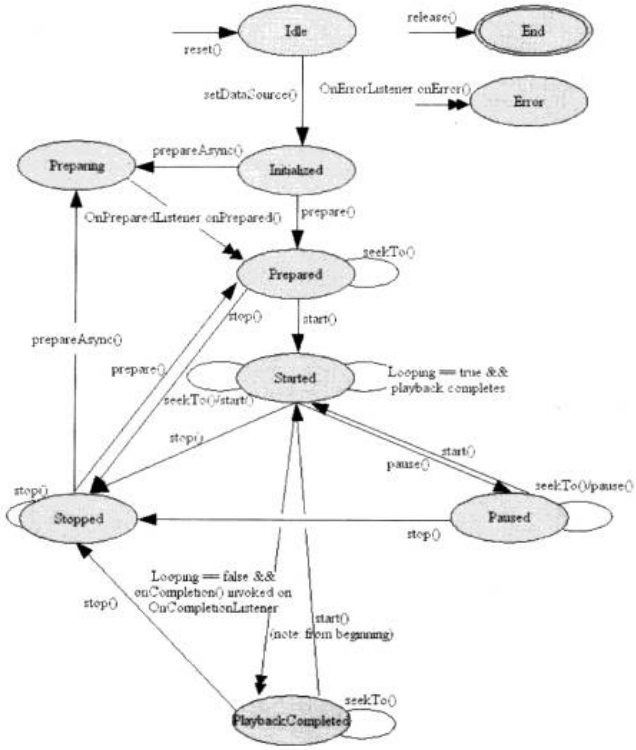


图 11.1 MediaPlayer 的状态图

- NoiseSuppressor：噪音压制器。
- BassBoost：重低音控制器。
- Equalizer：均衡控制器。
- PresetReverb：预设音场控制器。
- Virtualizer：示波器。

上面的子类中前三个子类的用法很简单，只要调用它们的静态 create()方法创建相应的实例，再调用它们的 isAvailable()判断是否可用，再调用 setEnabled(boolean enabled)方法启用相应效果即可。

下面是启用取消回声的示意代码：

```
// 获取取消回声控制器
AcousticEchoCanceler canceler = AcousticEchoCanceler.create(0
      , mPlayer.getAudioSessionId())
if(canceler.isAvailable())
{
      // 启用取消回声功能
      canceler.setEnabled(true);
}
```

下面是启用自动增益的示意代码：

```
// 获取自动增益控制器
AutomaticGainControl ctrl = AutomaticGainControl.create(0
      , mPlayer.getAudioSessionId())
if(ctrl.isAvailable())
{
      // 启用自动增益功能
      ctrl.setEnabled(true);
```

```
}
```

下面是启用噪音压制的示意代码：

```
// 获取噪音压制控制器
NoiseSuppressor suppressor = NoiseSuppressor.create(0
      , mPlayer.getAudioSessionId())
if(suppressor.isAvailable())
{
      // 启用噪音压制功能
      suppressor.setEnabled(true);
}
```

BassBoost、Equalizer、PresetReverb、Virtualizer 这 4 个类，都需要调用构造器来创建实例。创建实例时，同样需要传入一个 audioSession 参数，为了启用它们，同样需要调用 AudioEffect 基类的 setEnabled(true)方法。

获取 BassBoost 对象之后，可调用它的 setStrength(short strength)方法来设置重低音的强度。

获取 PresetReverb 对象之后，可调用它的 setPreset(short preset)设置使用预设置的音场。Equalizer 提供了 getNumberOfPresets()获取系统所有预设的音场，并提供了 getPresetName()获取预设音场名称。

获取 Equalizer 对象之后，可调用它的 getNumberOfBands()方法获取该均衡器支持的总频率数，再调用 getCenterFreq(short band)方法根据索引来获取频率。当用户想为某个频率的均衡器设置参数值时，可调用 setBandLevel(short band, short level)方法进行设置。

Virtualizer 对象并不用于控制音乐播放效果，它只是显示音乐的播放波形。为了实时显示该示波器的数据，需要为该组件设置一个 OnDataCaptureListener 监听器，该监听器将负责更新波形显示组件的界面。

下面用一个实例来示范上面 4 个特效类的用法。

实例：音乐的示波器、均衡、重低音和音场

下面的实例无需界面布局文件，使用一个 LinearLayout 容器来盛装一个示波器 View 组件，该示波器 View 组件将负责绘制 Virtualizer 传过来的数据；LinearLayout 添加多个 SeekBar 来控制 Equalizer 支持的所有频率的均衡值；LinearLayout 还添加一个 SeekBar 来控制重低音的强度；LinearLayout 还添加一个 Spinner 让用户选择预设音场。

下面是该实例的 Activity 代码。

程序清单：codes\11\11.1\MediaPlayerTest\src\org\crazyit\sound\MediaPlayerTest.java

```
public class MediaPlayerTest extends Activity
{
      // 定义播放声音的 MediaPlayer
      private MediaPlayer mPlayer;
      // 定义系统的示波器
      private Visualizer mVisualizer;
      // 定义系统的均衡器
      private Equalizer mEqualizer;
      // 定义系统的重低音控制器
      private BassBoost mBass;
      // 定义系统的预设音场控制器
      private PresetReverb mPresetReverb;
      private LinearLayout layout;
```

```
private List<Short> reverbNames = new ArrayList<Short>();
private List<String> reverbVals = new ArrayList<String>();
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    // 设置控制音乐声音
    setVolumeControlStream(AudioManager.STREAM_MUSIC);
    layout = new LinearLayout(this);
    layout.setOrientation(LinearLayout.VERTICAL);
    setContentView(layout);
    // 创建 MediaPlayer 对象
    mPlayer = MediaPlayer.create(this, R.raw.beautiful);
    // 初始化示波器
    setupVisualizer();
    // 初始化均衡控制器
    setupEqualizer();
    // 初始化重低音控制器
    setupBassBoost();
    // 初始化预设音场控制器
    setupPresetReverb();
    // 开发播放音乐
    mPlayer.start();
}
private void setupVisualizer()
{
    // 创建 MyVisualizerView 组件，用于显示波形图
    final MyVisualizerView mVisualizerView =
        new MyVisualizerView(this);
    mVisualizerView.setLayoutParams(new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        (int) (120f * getResources().getDisplayMetrics().density)));
    // 将 MyVisualizerView 组件添加到 layout 容器中
    layout.addView(mVisualizerView);
    // 以 MediaPlayer 的 AudioSessionId 创建 Visualizer
    // 相当于设置 Visualizer 负责显示该 MediaPlayer 的音频数据
    mVisualizer = new Visualizer(mPlayer.getAudioSessionId());
    mVisualizer.setCaptureSize(Visualizer.getCaptureSizeRange()[1]);
    // 为 mVisualizer 设置监听器
    mVisualizer.setDataCaptureListener(
        new Visualizer.OnDataCaptureListener()
        {
            @Override
            public void onFftDataCapture(Visualizer visualizer,
                byte[] fft, int samplingRate)
            {
            }
            @Override
            public void onWaveFormDataCapture(Visualizer visualizer,
                byte[] waveform, int samplingRate)
            {
                // 用 waveform 波形数据更新 mVisualizerView 组件
                mVisualizerView.updateVisualizer(waveform);
            }
        }, Visualizer.getMaxCaptureRate() / 2, true, false);
    mVisualizer.setEnabled(true);
}
// 初始化均衡控制器的方法
private void setupEqualizer()
{
    // 以 MediaPlayer 的 AudioSessionId 创建 Equalizer
```

```
// 相当于设置 Equalizer 负责控制该 MediaPlayer
mEqualizer = new Equalizer(0, mPlayer.getAudioSessionId());
// 启用均衡控制效果
mEqualizer.setEnabled(true);
TextView eqTitle = new TextView(this);
eqTitle.setText("均衡器：");
layout.addView(eqTitle);
// 获取均衡控制器支持最小值和最大值
final short minEQLevel = mEqualizer.getBandLevelRange()[0];
short maxEQLevel = mEqualizer.getBandLevelRange()[1];
// 获取均衡控制器支持的所有频率
short brands = mEqualizer.getNumberOfBands();
for (short i = 0; i < brands; i++)
{
    TextView eqTextView = new TextView(this);
    // 创建一个 TextView，用于显示频率
    eqTextView.setLayoutParams(new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        ViewGroup.LayoutParams.WRAP_CONTENT));
    eqTextView.setGravity(Gravity.CENTER_HORIZONTAL);
    // 设置该均衡控制器的频率
    eqTextView.setText((mEqualizer.getCenterFreq(i) / 1000)
        + " Hz");
    layout.addView(eqTextView);
    // 创建一个水平排列组件的 LinearLayout
    LinearLayout tmpLayout = new LinearLayout(this);
    tmpLayout.setOrientation(LinearLayout.HORIZONTAL);
    // 创建显示均衡控制器最小值的 TextView
    TextView minDbTextView = new TextView(this);
    minDbTextView.setLayoutParams(new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.WRAP_CONTENT,
        ViewGroup.LayoutParams.WRAP_CONTENT));
    // 显示均衡控制器的最小值
    minDbTextView.setText((minEQLevel / 100) + " dB");
    // 创建显示均衡控制器最大值的 TextView
    TextView maxDbTextView = new TextView(this);
    maxDbTextView.setLayoutParams(new ViewGroup.LayoutParams(
        ViewGroup.LayoutParams.WRAP_CONTENT,
        ViewGroup.LayoutParams.WRAP_CONTENT));
    // 显示均衡控制器的最大值
    maxDbTextView.setText((maxEQLevel / 100) + " dB");
    LinearLayout.LayoutParams layoutParams = new
        LinearLayout.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        ViewGroup.LayoutParams.WRAP_CONTENT);
    layoutParams.weight = 1;
    // 定义 SeekBar 作为调整工具
    SeekBar bar = new SeekBar(this);
    bar.setLayoutParams(layoutParams);
    bar.setMax(maxEQLevel - minEQLevel);
    bar.setProgress(mEqualizer.getBandLevel(i));
    final short brand = i;
    // 为 SeekBar 的拖动事件设置事件监听器
    bar.setOnSeekBarChangeListener(new SeekBar
        .OnSeekBarChangeListener()
    {
        @Override
        public void onProgressChanged(SeekBar seekBar,
            int progress, boolean fromUser)
        {
            // 设置该频率的均衡值
```

```
                        mEqualizer.setBandLevel(brand,
                            (short) (progress + minEQLevel));
                    }
                    @Override
                    public void onStartTrackingTouch(SeekBar seekBar)
                    {
                    }
                    @Override
                    public void onStopTrackingTouch(SeekBar seekBar)
                    {
                    }
                });
                // 使用水平排列组件的 LinearLayout "盛装" 三个组件
                tmpLayout.addView(minDbTextView);
                tmpLayout.addView(bar);
                tmpLayout.addView(maxDbTextView);
                // 将水平排列组件的 LinearLayout 添加到 myLayout 容器中
                layout.addView(tmpLayout);
        }
    }
    // 初始化重低音控制器
    private void setupBassBoost()
    {
        // 以 MediaPlayer 的 AudioSessionId 创建 BassBoost
        // 相当于设置 BassBoost 负责控制该 MediaPlayer
        mBass = new BassBoost(0, mPlayer.getAudioSessionId());
        // 设置启用重低音效果
        mBass.setEnabled(true);
        TextView bbTitle = new TextView(this);
        bbTitle.setText("重低音: ");
        layout.addView(bbTitle);
        // 使用 SeekBar 作为重低音的调整工具
        SeekBar bar = new SeekBar(this);
        // 重低音的范围为 0～1000
        bar.setMax(1000);
        bar.setProgress(0);
        // 为 SeekBar 的拖动事件设置事件监听器
        bar.setOnSeekBarChangeListener(new SeekBar
            .OnSeekBarChangeListener()
        {
            @Override
            public void onProgressChanged(SeekBar seekBar
                , int progress, boolean fromUser)
            {
                // 设置重低音的强度
                mBass.setStrength((short) progress);
            }
            @Override
            public void onStartTrackingTouch(SeekBar seekBar)
            {
            }
            @Override
            public void onStopTrackingTouch(SeekBar seekBar)
            {
            }
        });
        layout.addView(bar);
    }
    // 初始化预设音场控制器
    private void setupPresetReverb()
    {
```

```
        // 以MediaPlayer的AudioSessionId创建PresetReverb
        // 相当于设置PresetReverb负责控制该MediaPlayer
        mPresetReverb = new PresetReverb(0,
            mPlayer.getAudioSessionId());
        // 设置启用预设音场控制
        mPresetReverb.setEnabled(true);
        TextView prTitle = new TextView(this);
        prTitle.setText("音场");
        layout.addView(prTitle);
        // 获取系统支持的所有预设音场
        for (short i = 0; i < mEqualizer.getNumberOfPresets(); i++)
        {
            reverbNames.add(i);
            reverbVals.add(mEqualizer.getPresetName(i));
        }
        // 使用Spinner作为音场选择工具
        Spinner sp = new Spinner(this);
        sp.setAdapter(new ArrayAdapter<String>(MediaPlayerTest.this,
            android.R.layout.simple_spinner_item, reverbVals));
        // 为Spinner的列表项选中事件设置监听器
        sp.setOnItemSelectedListener(new Spinner
            .OnItemSelectedListener()
        {
            @Override
            public void onItemSelected(AdapterView<?> arg0
                , View arg1, int arg2, long arg3)
            {
                // 设定音场
                mPresetReverb.setPreset(reverbNames.get(arg2));
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
            }
        });
        layout.addView(sp);
    }
    @Override
    protected void onPause()
    {
        super.onPause();
        if (isFinishing() && mPlayer != null)
        {
            // 释放所有对象
            mVisualizer.release();
            mEqualizer.release();
            mPresetReverb.release();
            mBass.release();
            mPlayer.release();
            mPlayer = null;
        }
    }
    private static class MyVisualizerView extends View
    {
        // bytes数组保存了波形抽样点的值
        private byte[] bytes;
        private float[] points;
        private Paint paint = new Paint();
        private Rect rect = new Rect();
        private byte type = 0;
        public MyVisualizerView(Context context)
```

```
{
        super(context);
        bytes = null;
        // 设置画笔的属性
        paint.setStrokeWidth(1f);
        paint.setAntiAlias(true);
        paint.setColor(Color.GREEN);
        paint.setStyle(Style.FILL);
}
public void updateVisualizer(byte[] ftt)
{
        bytes = ftt;
        // 通知该组件重绘自己
        invalidate();
}

@Override
public boolean onTouchEvent(MotionEvent me)
{
        // 当用户触碰该组件时，切换波形类型
        if(me.getAction() != MotionEvent.ACTION_DOWN)
        {
                return false;
        }
        type ++;
        if(type >= 3)
        {
                type = 0;
        }
        return true;
}
@Override
protected void onDraw(Canvas canvas)
{
        super.onDraw(canvas);
        if (bytes == null)
        {
                return;
        }
        // 绘制白色背景（主要为了印刷时好看）
        canvas.drawColor(Color.WHITE);
        // 使用 rect 对象记录该组件的宽度和高度
        rect.set(0,0,getWidth(),getHeight());
        switch(type)
        {
                // -------绘制块状的波形图-------
                case 0:
                        for (int i = 0; i < bytes.length - 1; i++)
                        {
                                float left = getWidth() * i / (bytes.length - 1);
                                // 根据波形值计算该矩形的高度
                                float top = rect.height()-(byte)(bytes[i+1]+128)
                                        * rect.height() / 128;
                                float right = left + 1;
                                float bottom = rect.height();
                                canvas.drawRect(left, top, right, bottom, paint);
                        }
                        break;
                // -------绘制柱状的波形图（每隔 18 个抽样点绘制一个矩形）-------
                case 1:
                        for (int i = 0; i < bytes.length - 1; i += 18)
```

```
                {
                    float left = rect.width()*i/(bytes.length - 1);
                    // 根据波形值计算该矩形的高度
                    float top = rect.height()-(byte)(bytes[i+1]+128)
                        * rect.height() / 128;
                    float right = left + 6;
                    float bottom = rect.height();
                    canvas.drawRect(left, top, right, bottom, paint);
                }
                break;
            // -------绘制曲线波形图-------
            case 2:
                // 如果 point 数组还未初始化
                if (points == null || points.length < bytes.length * 4)
                {
                    points = new float[bytes.length * 4];
                }
                for (int i = 0; i < bytes.length - 1; i++)
                {
                    // 计算第 i 个点的 x 坐标
                    points[i * 4] = rect.width()*i/(bytes.length - 1);
                    // 根据 bytes[i]的值（波形点的值）计算第 i 个点的 y 坐标
                    points[i * 4 + 1] = (rect.height() / 2)
                        + ((byte) (bytes[i] + 128)) * 128
                        / (rect.height() / 2);
                    // 计算第 i+1 个点的 x 坐标
                    points[i * 4 + 2] = rect.width() * (i + 1)
                        / (bytes.length - 1);
                    // 根据 bytes[i+1]的值（波形点的值）计算第 i+1 个点的 y 坐标
                    points[i * 4 + 3] = (rect.height() / 2)
                        + ((byte) (bytes[i + 1] + 128)) * 128
                        / (rect.height() / 2);
                }
                // 绘制波形曲线
                canvas.drawLines(points, paint);
                break;
            }
        }
    }
}
```

上面的程序中定义了 4 个方法：setupVisualizer()、setupEqualizer()、setupBassBoost()和 setupPresetReverb()，这 4 个方法分别创建 Visualizer、Equalizer、BassBoost 和 PresetReverb 对象，并调用它们的方法控制音乐特效。

上面的实例中包含了一个 MyVisualizerView 内部类，该内部类会根据 Visualizer 传过来的数据动态绘制波形效果。本例提供了三种波形：块状波形、柱状波形和曲线波形——当用户单击 MyVisualizerView 组件时将会切换波形。

运行该实例，可以看到如图 11.2 所示效果。

如果用户单击该程序最上方的示波器组件，该组件将会切换一种波形显示。单击下方的音场，将可以看到系统支持的所有预设音场，如图 11.3 所示。

➤➤11.1.3　使用 SoundPool 播放音效

如果应用程序经常需要播放密集、短促的音效，这时还用 MediaPlayer 就显得有些不合适了。MediaPlayer 存在如下缺点：

- 资源占用量较高、延迟时间较长。
- 不支持多个音频同时播放。

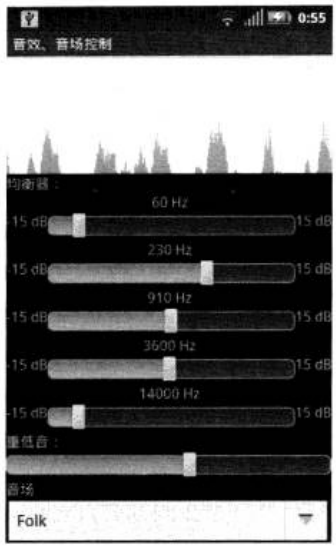


图 11.2 示波器、均衡、重低音和音场

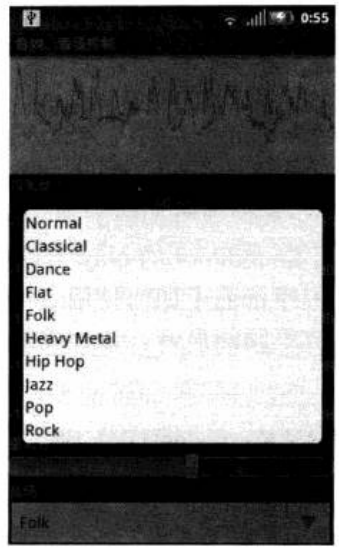


图 11.3 曲线示波器和全部预设音场

除了前面介绍的 MediaPlayer 播放音频之外，Android 还提供了 SoundPool 来播放音效，SoundPool 使用音效池的概念来管理多个短促的音效，例如它可以开始就加载 20 个音效，以后在程序中按音效的 ID 进行播放。

SoundPool 主要用于播放一些较短的声音片段，与 MediaPlayer 相比，SoundPool 的优势在于 CPU 资源占用量低和反应延迟小。另外，SoundPool 还支持自行设置声音的品质、音量、播放比率等参数。

SoundPool 提供了一个构造器，该构造器可以指定它总共支持多少个声音（也就是池的大小）、声音的品质等。构造器如下。

- SoundPool(int maxStreams, int streamType, int srcQuality)：第一个参数指定支持多少个声音；第二个参数指定声音类型；第三个参数指定声音品质。

一旦得到了 SoundPool 对象之后，接下来就可调用 SoundPool 的多个重载的 load 方法来加载声音了，SoundPool 提供了如下 4 个 load 方法。

- int load(Context context, int resId, int priority)：从 resId 所对应的资源加载声音。
- int load(FileDescriptor fd, long offset, long length, int priority)：加载 fd 所对应的文件的 offset 开始、长度为 length 的声音。
- int load(AssetFileDescriptor afd, int priority)：从 afd 所对应的文件中加载声音。
- int load(String path, int priority)：从 path 对应的文件去加载声音。

上面 4 个方法中都有一个 priority 参数，该参数目前还没有任何作用，Android 建议将该参数设为 1，保持和未来的兼容性。

上面 4 个方法加载声音之后，都会返回该声音的的 ID，以后程序就可以通过该声音的 ID 来播放指定声音，SoundPool 提供的播放指定声音的方法如下。

- int play(int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)：该方法的第一个参数指定播放哪个声音；leftVolume、rightVolume 指定左、右的音量；priority 指定播放声音的优先级，数值越大，优先级越高；loop 指定是否

循环，0 为不循环，-1 为循环；rate 指定播放的比率，数值可从 0.5 到 2，1 为正常比率。

为了更好地管理 SoundPool 所加载的每个声音的 ID，程序一般会使用一个 HashMap<Integer,Integer>对象来管理声音。

归纳起来，使用 SoundPool 播放声音的步骤如下。

① 调用 SoundPool 的构造器创建 SoundPool 的对象。

② 调用 SoundPool 对象的 load()方法从指定资源、文件中加载声音。最好使用 HashMap<Integer,Integer>来管理所加载的声音。

③ 调用 SoundPool 的 play 方法播放声音。

下面的程序示范了如何使用 SoundPool 来播放音效，该程序提供了三个按钮，三个按钮分别用于播放不同的声音。该程序的界面十分简单，故此处不再给出界面布局代码，程序代码如下。

程序清单：codes\11\11.1\SoundPoolTest\src\org\crazyit\sound\SoundPoolTest.java

```
public class SoundPoolTest extends Activity implements OnClickListener
{
    Button bomb, shot, arrow;
    // 定义一个 SoundPool
    SoundPool soundPool;
    HashMap<Integer, Integer> soundMap =
        new HashMap<Integer, Integer>();
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bomb = (Button) findViewById(R.id.bomb);
        shot = (Button) findViewById(R.id.shot);
        arrow = (Button) findViewById(R.id.arrow);
        // 设置最多可容纳 10 个音频流，音频的品质为 5
        soundPool = new SoundPool(10
            , AudioManager.STREAM_SYSTEM, 5); //①
        // load 方法加载指定音频文件，并返回所加载的音频 ID
        // 此处使用 HashMap 来管理这些音频流
        soundMap.put(1, soundPool.load(this, R.raw.bomb, 1)); //②
        soundMap.put(2, soundPool.load(this, R.raw.shot, 1));
        soundMap.put(3, soundPool.load(this, R.raw.arrow, 1));
        bomb.setOnClickListener(this);
        shot.setOnClickListener(this);
        arrow.setOnClickListener(this);
    }
    // 重写 OnClickListener 监听器接口的方法
    @Override
    public void onClick(View source)
    {
        // 判断哪个按钮被单击
        switch (source.getId())
        {
            case R.id.bomb:
                soundPool.play(soundMap.get(1), 1, 1, 0, 0, 1); //③
                break;
            case R.id.shot:
                soundPool.play(soundMap.get(2), 1, 1, 0, 0, 1);
                break;
            case R.id.arrow:
```

```
                    soundPool.play(soundMap.get(3), 1, 1, 0, 0, 1);
                    break;
                }
            }
        }
```

上面的程序中①号粗体字代码用于创建 SoundPool 对象；②号粗体字代码用于使用 SoundPool 加载多个不同的声音；③号粗体字代码则用于根据声音 ID 来播放指定声音。这就是使用 SoundPool 播放声音的标准过程。

实际使用 SoundPool 播放声音时有如下几点需要注意：SoundPool 虽然可以一次性加载多个声音，但由于内存限制，因此应该避免使用 SoundPool 来播放歌曲或者做游戏背景音乐，只有那些短促、密集的声音才考虑使用 SoundPool 进行播放。

虽然 SoundPool 比 MediaPlayer 的效率好，但也不是绝对不存在延迟问题，尤其在那些性能不太好的手机中，SoundPool 的延迟问题会更严重。

11.1.4 使用 VideoView 播放视频

为了在 Android 应用中播放声音，Android 提供了 VideoView 组件，它就是一个位于 android.widget 包下的组件，它的作用与 ImageView 类似，只是 ImageView 用于显示图片，而 VideoView 用于播放视频。

使用 VideoView 播放视频的步骤如下。

① 在界面布局文件中定义 VideoView 组件，或在程序中创建 VideoView 组件。

② 调用 VideoView 的如下两个方法来加载指定视频。

- setVideoPath(String path)：加载 path 文件所代表的视频。
- setVideoURI(Uri uri)：加载 uri 所对应的视频。

③ 调用 VideoView 的 start()、stop()、pause()方法来控制视频播放。

实际上与 VideoView 一起结合使用的还有一个 MediaController 类，它的作用是提供一个友好的图形控制界面，通过该控制界面来控制视频的播放。

下面的程序示范了如何使用 VideoView 来播放视频。该程序提供了一个简单的界面，界面布局的代码如下。

程序清单：codes\11\11.1\VideoViewTest\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<!-- 定义 VideoView 播放视频 -->
<VideoView
      android:id="@+id/video"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
</LinearLayout>
```

上面的界面布局中定义了一个 VideoView 组件，接下来就可以在程序中使用该组件来播放视频了。播放视频时还结合了 MediaController 来控制视频的播放，该程序代码如下。

程序清单：codes\11\11.1\VideoViewTest\src\org\crazyit\sound\VideoViewTest.java

```
public class VedioViewTest extends Activity
{
```

```
VideoView videoView;
MediaController mController;
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    getWindow().setFormat(PixelFormat.TRANSLUCENT);
    setContentView(R.layout.main);
    // 获取界面上 VideoView 组件
    videoView = (VideoView) findViewById(R.id.video);
    // 创建 MediaController 对象
    mController = new MediaController(this);
    File video = new File("/mnt/sdcard/movie.mp4");
    if(video.exists())
    {
        videoView.setVideoPath(video.getAbsolutePath()); //①
        // 设置 videoView 与 mController 建立关联
        videoView.setMediaController(mController); //②
        // 设置 mController 与 videoView 建立关联
        mController.setMediaPlayer(videoView); //③
        // 让 VideoView 获取焦点
        videoView.requestFocus();
    }
}
}
```

上面的程序中①号粗体字代码用于让 VideoView 加载指定的视频文件，接下来该 VideoView 就可以用于播放该视频了；接下来②、③号代码用于建立 VideoView 与 MediaController 之间的关联，这样就不需要开发者自己去控制视频的播放、暂停等了，让 MediaController 进行控制即可。



图 11.4　使用 VideoPlayer 播放视频

运行该程序，保证/mnt/sdcard/movie.mp4 视频文件存在的前提下，将可以看到如图 11.4 所示的界面。

图 11.4 所示界面中快进键、暂停键、后退键，以及播放进度条就是由 MediaPlayerController 所提供的。



提示：运行该程序可能会遇到一些问题，如果读者使用了一些非标准的 MP4、3GP 文件，那么该应用程序将无法播放；因此建议读者自行使用手机录制一段兼容各种手机的、标准的 MP4、3GP 视频文件。

11.1.5　使用 MediaPlayer 和 SurfaceView 播放视频

使用 VideoView 播放视频简单、方便，但有些早期的开发者还是更喜欢使用 MediaPlayer 来播放视频，但由于 MediaPlayer 主要用于播放音频，因此它没有提供图像输出界面，此时就需要借助于 SurfaceView 来显示 MediaPlayer 播放的图像输出。

使用 MediaPlayer 播放视频的步骤如下。

① 创建 MediaPlayer 对象，并让它加载指定的视频文件。

② 在界面布局文件中定义 SurfaceView 组件，或在程序中创建 SurfaceView 组件。并为 SurfaceView 的 SurfaceHolder 添加 Callback 监听器。

③ 调用 MediaPlayer 对象的 setDisplay(SurfaceHolder sh)将所播放的视频图像输出到指定的 SurfaceView 组件。

④ 调用 MediaPlayer 对象的 start()、stop()和 pause()方法控制视频的播放。

下面的程序使用了 MediaPlayer 和 SurfaceView 来播放视频，并为该程序提供了三个按钮来控制视频的播放、暂停和停止。该程序的界面布局比较简单，故此处不再给出。该程序的代码如下。

程序清单：codes\11\11.1\SurfaceViewPlayVideo\src\org\crazyit\sound\SurfaceViewPlayVideo. java

```
public class SurfaceViewPlayVideo extends Activity
    implements OnClickListener
{
    SurfaceView surfaceView;
    ImageButton play, pause, stop;
    MediaPlayer mPlayer;
    // 记录当前视频的播放位置
    int position;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面中的三个按钮
        play = (ImageButton) findViewById(R.id.play);
        pause = (ImageButton) findViewById(R.id.pause);
        stop = (ImageButton) findViewById(R.id.stop);
        // 为三个按钮的单击事件绑定事件监听器
        play.setOnClickListener(this);
        pause.setOnClickListener(this);
        stop.setOnClickListener(this);
        // 创建 MediaPlayer
        mPlayer = new MediaPlayer();
        surfaceView = (SurfaceView) this.findViewById(R.id.surfaceView);
        // 设置播放时打开屏幕
        surfaceView.getHolder().setKeepScreenOn(true);
        surfaceView.getHolder().addCallback(new SurfaceListener());
    }
    @Override
    public void onClick(View source)
    {
        try
        {
            switch (source.getId())
            {
                // 播放按钮被单击
                case R.id.play:
                    play();
                    break;
                // 暂停按钮被单击
                case R.id.pause:
                    if (mPlayer.isPlaying())
                    {
                        mPlayer.pause();
                    }
                    else
                    {
                        mPlayer.start();
                    }
```

```
                    break;
                // 停止按钮被单击
                case R.id.stop:
                    if (mPlayer.isPlaying()) mPlayer.stop();
                    break;
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
    private void play() throws IOException
    {
        mPlayer.reset();
        mPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
        // 设置需要播放的视频
        mPlayer.setDataSource("/mnt/sdcard/movie.3gp");
        // 把视频画面输出到 SurfaceView
        mPlayer.setDisplay(surfaceView.getHolder()); //①
        mPlayer.prepare();
        // 获取窗口管理器
        WindowManager wManager = getWindowManager();
        DisplayMetrics metrics = new DisplayMetrics();
        // 获取屏幕大小
        wManager.getDefaultDisplay().getMetrics(metrics);
        // 设置视频保持纵横比缩放到占满整个屏幕
        surfaceView.setLayoutParams(new LayoutParams(metrics.widthPixels
            , mPlayer.getVideoHeight() * metrics.widthPixels
            / mPlayer.getVideoWidth()));
        mPlayer.start();
    }
    private class SurfaceListener implements SurfaceHolder.Callback
    {
        @Override
        public void surfaceChanged(SurfaceHolder holder, int format,
            int width, int height)
        {
        }
        @Override
        public void surfaceCreated(SurfaceHolder holder)
        {
            if (position > 0)
            {
                try
                {
                    // 开始播放
                    play();
                    // 并直接从指定位置开始播放
                    mPlayer.seekTo(position);
                    position = 0;
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        }
        @Override
        public void surfaceDestroyed(SurfaceHolder holder)
        {
```

```
        }
    }
    // 当其他 Activity 被打开时，暂停播放
    @Override
    protected void onPause()
    {
        if (mPlayer.isPlaying())
        {
            // 保存当前的播放位置
            position = mPlayer.getCurrentPosition();
            mPlayer.stop();
        }
        super.onPause();
    }
    @Override
    protected void onDestroy()
    {
        // 停止播放
        if (mPlayer.isPlaying()) mPlayer.stop();
        // 释放资源
        mPlayer.release();
        super.onDestroy();
    }
}
```

从上面的代码不难看出，使用 MediaPlayer 播放视频与播放音频的步骤大同小异，关键的区别在于①号粗体字代码，设置使用 SurfaceView 来显示 MediaPlayer 播放时的图像输出。当然，由于程序需要使用 SurfaceView 来显示 MediaPlayer 的图像输出，因此程序需要一些代码来维护 SurfaceView、SurfaceHolder 对象。

运行上面的程序，将可以看到如图 11.5 所示的播放界面。

从上面的开发过程不难看出，使用 MediaPlayer 播放视频要复杂一些，而且需要自己开发控制按钮来控制视频播放。因此一般推荐使用 VideoView 来播放视频。



图 11.5 使用 MediaPlayer 播放视频

11.2 使用 MediaRecorder 录制音频

手机一般都提供了麦克风硬件，而 Android 系统就可以利用该硬件来录制音频了。

为了在 Android 应用中录制音频，Android 提供了 MediaRecorder 类。使用 MediaRecorder 录制音频的过程很简单，按如下步骤进行即可。

① 创建 MediaRecorder 对象。

② 调用 MediaRecorder 对象的 setAudioSource()方法设置声音来源，一般传入 MediaRecorder. AudioSource.MIC 参数指定录制来自麦克风的声音。

③ 调用 MediaRecorder 对象的 setOutputFormat()设置所录制的音频文件的格式。

④ 调用 MediaRecorder 对象的 setAudioEncoder()、setAudioEncodingBitRate(int bitRate)、setAudioSamplingRate(int samplingRate)设置所录制的声音的编码格式、编码位率、采样率等，这些参数将可以控制所录制的声音的品质、文件的大小。一般来说，声音品质越好，声音文件越大。

⑤ 调用 MediaRecorder 的 setOutputFile(String path)方法设置录制的音频文件的保存

位置。

⑥ 调用 MediaRecorder 的 prepare()方法准备录制。

⑦ 调用 MediaRecorder 对象的 start()方法开始录制。

⑧ 录制完成，调用 MediaRecorder 对象的 stop()方法停止录制，并调用 release()方法释放资源。

注意：

上面的步骤中第 3 和第 4 两个步骤千万不能搞反，否则程序将会抛出 IllegalStateException 异常。



图 11.6 显示了 MediaPlayer 的状态图。

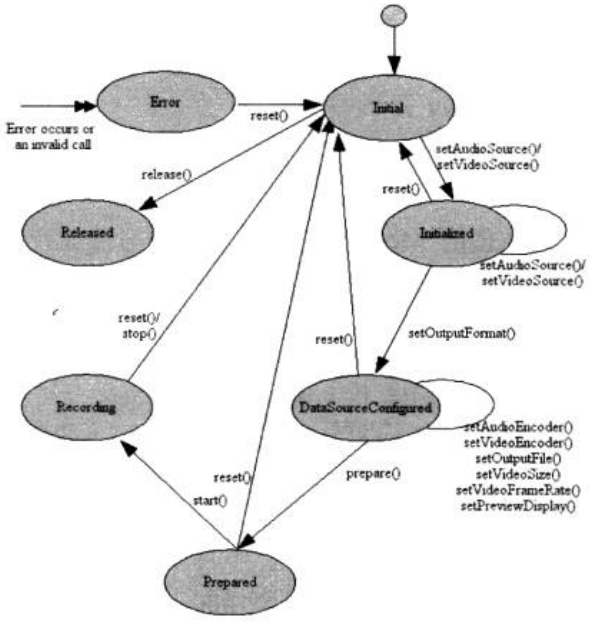


图 11.6　MediaRecorder 的状态图

实例：录制音乐

下面的程序示范了如何使用 MediaRecorder 来录制声音，该程序的界面布局很简单，只提供了两个简单的按钮来控制录音开始、停止，故此处不再给出界面布局文件，程序代码如下。

程序清单：codes\11\11.2\RecordSound\src\org\crazyit\sound\RecordSound.java

```
public class RecordSound extends Activity
    implements OnClickListener
{
    // 定义界面上的两个按钮
    ImageButton record, stop;
    // 系统的音频文件
    File soundFile;
    MediaRecorder mRecorder;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的两个按钮
        record = (ImageButton) findViewById(R.id.record);
        stop = (ImageButton) findViewById(R.id.stop);
        // 为两个按钮的单击事件绑定监听器
        record.setOnClickListener(this);
        stop.setOnClickListener(this);
    }
    @Override
    public void onDestroy()
    {
        if (soundFile != null && soundFile.exists())
        {
            // 停止录音
            mRecorder.stop();
            // 释放资源
            mRecorder.release();
            mRecorder = null;
        }
        super.onDestroy();
    }
    @Override
    public void onClick(View source)
    {
        switch (source.getId())
        {
        // 单击录音按钮
            case R.id.record:
                if (!Environment.getExternalStorageState().equals(
                        android.os.Environment.MEDIA_MOUNTED))
                {
                    Toast.makeText(RecordSound.this, "SD卡不存在，请插入SD卡！",
                            Toast.LENGTH_SHORT).show();
                    return;
                }
                try
                {
                    // 创建保存录音的音频文件
                    soundFile = new File(Environment
                            .getExternalStorageDirectory().getCanonicalFile()
                            + "/sound.amr");
                    mRecorder = new MediaRecorder();
                    // 设置录音的声音来源
                    mRecorder.setAudioSource(MediaRecorder
                            .AudioSource.MIC);
                    // 设置录制的声音的输出格式（必须在设置声音编码格式之前设置）
                    mRecorder.setOutputFormat(MediaRecorder
                            .OutputFormat.THREE_GPP);
                    // 设置声音编码的格式
                    mRecorder.setAudioEncoder(MediaRecorder
                            .AudioEncoder.AMR_NB);
                    mRecorder.setOutputFile(soundFile.getAbsolutePath());
                    mRecorder.prepare();
                    // 开始录音
                    mRecorder.start(); //①
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
```

```
                    break;
                // 单击停止按钮
                case R.id.stop:
                    if (soundFile != null && soundFile.exists())
                    {
                        // 停止录音
                        mRecorder.stop();  //②
                        // 释放资源
                        mRecorder.release();  //③
                        mRecorder = null;
                    }
                    break;
            }
        }
    }
```

上面的程序中大段的粗体字代码用于设置录音的相关参数，比如输出文件的格式、声音来源等。上面的程序中①粗体字代码控制 MediaRecorder 开始录音；当用户单击“停止录制”按钮时，程序中②号代码控制 MediaRecorder 停止录音，③号粗体字代码用于释放资源。

运行该程序，将看到如图 11.7 所示的界面。

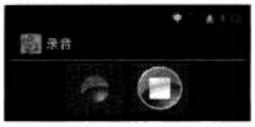


图 11.7　录音界面

单击图 11.7 所示程序中第一个按钮开始录音，单击第二个按钮则可结束录音。录音完成后将可以看到/mnt/sdcard/目录下生成一个 sound.amr 文件，这就是刚刚录制的音频文件——Android 模拟器将会直接使用宿主电脑上的麦克风，因此如果读者的电脑上有麦克风，那么该程序即可正常录制声音。

上面的程序需要使用系统的麦克风进行录音，因此需要向该程序授予录音的权限；除此之外，还需要授予该程序向外部存储设备写入数据的权限。也就是在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 授予该程序录制声音的权限 -->
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<!-- 授予该程序向外部存储器写入数据的权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

11.3　控制摄像头拍照

现在的手机一般都会提供相机功能，有些相机的镜头甚至支持 1000 万以上像素，有些甚至支持光学变焦，这些手机已经变成了专业数码相机。为了充分利用手机上的相机功能，Android 应用可以控制拍照和录制视频。

➤➤11.3.1　通过 Camera 进行拍照

Android 应用提供了 Camera 来控制拍照，使用 Camera 进行拍照也比较简单，按如下步骤进行即可。

① 调用 Camera 的 open()方法打开相机。该方法默认打开后置摄像头。如果需要打开指定摄像头，可以为该方法传入摄像头 ID。

② 调用 Camera 的 getParameters()方法获取拍照参数。该方法返回一个 Camera.Parameters 对象。

③ 调用 Camera.Parameters 对象方法设置拍照参数。

④ 调用 Camera 的 startPreview()方法开始预览取景，在预览取景之前需要调用 Camera 的 setPreviewDisplay(SurfaceHolder holder)方法设置使用哪个 SurfaceView 来显示取景图片。

⑤ 调用 Camera 的 takePicture()方法进行拍照。

⑥ 结束程序时，调用 Camera 的 stopPreview()结束取景预览，并调用 release()方法释放资源。

实例：拍照时自动对焦

下面的实例示范了使用 Camera 来进行拍照、当用户按下拍照键时，该应用会自动对焦，当对焦成功时拍下照片。该程序的界面中只提供了一个 SurfaceView 来显示预览取景，十分简单。该程序代码如下。

程序清单：codes\11\11.3\CaptureImage\src\org\crazyit\sound\CaptureImage.java

```
public class CaptureImage extends Activity
{
    SurfaceView sView;
    SurfaceHolder surfaceHolder;
    int screenWidth, screenHeight;
    // 定义系统所用的照相机
    Camera camera;
    // 是否在预览中
    boolean isPreview = false;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 设置全屏
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.main);
        // 获取窗口管理器
        WindowManager wm = getWindowManager();
        Display display = wm.getDefaultDisplay();
        DisplayMetrics metrics = new DisplayMetrics();
        // 获取屏幕的宽和高
        display.getMetrics(metrics);
        screenWidth = metrics.widthPixels;
        screenHeight = metrics.heightPixels;
        // 获取界面中 SurfaceView 组件
        sView = (SurfaceView) findViewById(R.id.sView);
        // 设置该 Surface 不需要自己维护缓冲区
        sView.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        // 获得 SurfaceView 的 SurfaceHolder
        surfaceHolder = sView.getHolder();
        // 为 surfaceHolder 添加一个回调监听器
        surfaceHolder.addCallback(new Callback()
        {
            @Override
            public void surfaceChanged(SurfaceHolder holder, int format,
                int width, int height)
            {
            }
            @Override
            public void surfaceCreated(SurfaceHolder holder)
```

```
            {
                // 打开摄像头
                initCamera();
            }
            @Override
            public void surfaceDestroyed(SurfaceHolder holder)
            {
                // 如果 camera 不为 null，释放摄像头
                if (camera != null)
                {
                    if (isPreview) camera.stopPreview();
                    camera.release();
                    camera = null;
                }
            }
        });
    }
    private void initCamera()
    {
        if (!isPreview)
        {
            // 此处默认打开后置摄像头
            // 通过传入参数可以打开前置摄像头
            camera = Camera.open(0);  //①
            camera.setDisplayOrientation(90);
        }
        if (camera != null && !isPreview)
        {
            try
            {
                Camera.Parameters parameters = camera.getParameters();
                // 设置预览照片的大小
                parameters.setPreviewSize(screenWidth, screenHeight);
                // 设置预览照片时每秒显示多少帧的最小值和最大值
                parameters.setPreviewFpsRange(4, 10);
                // 设置图片格式
                parameters.setPictureFormat(ImageFormat.JPEG);
                // 设置 JPG 照片的质量
                parameters.set("jpeg-quality", 85);
                // 设置照片的大小
                parameters.setPictureSize(screenWidth, screenHeight);
                // 通过 SurfaceView 显示取景画面
                camera.setPreviewDisplay(surfaceHolder);  //②
                // 开始预览
                camera.startPreview();  // ③
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            isPreview = true;
        }
    }
    public void capture(View source)
    {
        if (camera != null)
        {
            // 控制摄像头自动对焦后才拍照
            camera.autoFocus(autoFocusCallback);  //④
        }
    }
```

```
AutoFocusCallback autoFocusCallback = new AutoFocusCallback()
{
    // 当自动对焦时激发该方法
    @Override
    public void onAutoFocus(boolean success, Camera camera)
    {
        if (success)
        {
            // takePicture()方法需要传入三个监听器参数
            // 第一个监听器：当用户按下快门时激发该监听器
            // 第二个监听器：当相机获取原始照片时激发该监听器
            // 第三个监听器：当相机获取 JPG 照片时激发该监听器
            camera.takePicture(new ShutterCallback()
            {
                public void onShutter()
                {
                    // 按下快门瞬间会执行此处代码
                }
            }, new PictureCallback()
            {
                public void onPictureTaken(byte[] data, Camera c)
                {
                    // 此处代码可以决定是否需要保存原始照片信息
                }
            }, myJpegCallback); //⑤
        }
    }
};
PictureCallback myJpegCallback = new PictureCallback()
{
    @Override
    public void onPictureTaken(byte[] data, Camera camera)
    {
        // 根据拍照所得的数据创建位图
        final Bitmap bm = BitmapFactory.decodeByteArray(data, 0,
            data.length);
        // 加载/layout/save.xml 文件对应的布局资源
        View saveDialog = getLayoutInflater().inflate(R.layout.save,
            null);
        final EditText photoName = (EditText) saveDialog
            .findViewById(R.id.phone_name);
        // 获取 saveDialog 对话框上的 ImageView 组件
        ImageView show = (ImageView) saveDialog
            .findViewById(R.id.show);
        // 显示刚刚拍得的照片
        show.setImageBitmap(bm);
        // 使用对话框显示 saveDialog 组件
        new AlertDialog.Builder(CaptureImage.this).setView(saveDialog)
            .setPositiveButton("保存", new OnClickListener()
            {
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                    // 创建一个位于 SD 卡上的文件
                    File file = new File(Environment
                        .getExternalStorageDirectory(), photoName
                        .getText().toString() + ".jpg");
                    FileOutputStream outStream = null;
                    try
                    {
                        // 打开指定文件对应的输出流
```

```
                                        outStream = new FileOutputStream(file);
                                        // 把位图输出到指定文件中
                                        bm.compress(CompressFormat.JPEG, 100,
                                            outStream);
                                        outStream.close();
                                    }
                                    catch (IOException e)
                                    {
                                        e.printStackTrace();
                                    }
                                }
                            }).setNegativeButton("取消", null).show();
                        // 重新浏览
                        camera.stopPreview();
                        camera.startPreview();
                        isPreview = true;
                    }
                };
            }
```

上面的程序中①号粗体字代码用于打开系统摄像头，Camera 的 open()默认打开设备后置摄像头；如果需要打开设备指定摄像头（比如前置摄像头），可以在调用 open()方法时传入摄像头 ID。



提示：

Camera 提供了 getNumberOfCameras()和 getCameraInfo()方法来获取设备的摄像头数量和摄像头信息。例如如下代码：

```
// 获取设备上摄像头的数量
int cameraCount = Camera.getNumberOfCameras();
// 创建一个空的 CameraInfo 对象，用于获取摄像头信息
Camera.CameraInfo cameraInfo = new Camera.CameraInfo();
for ( int camIdx = 0; camIdx < cameraCount;camIdx++ )
{
    // 获取第 camIdx 摄像头的信息
    Camera.getCameraInfo( camIdx, cameraInfo);
    // 接下来的代码就可以通过 cameraInfo 来获取摄像头信息了
    ...
}
```

上面的程序中大段粗体字代码用于设置相机的拍照参数，这些参数将可以控制相片的品质和相片文件的大小。

上面的程序中②号粗体字代码设置使用指定的 SurfaceView 来显示取景预览图片，程序中③号粗体字代码则用于开始预览取景。当用户按下拍照键时，程序的④号粗体字代码调用了 autoFocus()方法控制自动对焦，并在对焦完成后拍照。对焦完成后，程序中⑤号代码调用 takePicture()方法进行拍照。



图 11.8　预览界面

调用 takePicture()方法进行拍照时需要传入三个参数，第三个参数是一个 PictureCallback 对象——当程序获取了拍照所得的图片数据之后，PictureCallback 对象将会被回调，该对象将可以负责对相片进行保存或上传至网络。

在 Android 模拟器上运行该程序可能看到如图 11.8 所示的预览界面，由于模拟器调用 Camera 的 open(0)方法无法打开任何摄像头——这是因为模拟器不会使用宿主电脑的摄像头作为相机镜头造成的。Android 模拟

器只能使用图 11.8 所示的“假”预览界面。



提示： 为了让模拟器能显示图 11.8 所示的预览界面，建议读者在创建 AVD 模拟添加摄像头支持，添加摄像头支持，可以在如图 1.7 所示的创建 AVD 对话框中将 Back Camera 设为 Emulated。

在模拟器上按下图 11.8 所示界面的“拍照”键没有任何效果，这是因为模拟器的连摄像头都是“假”的，自然也就无法自动对焦了——而本实例要求自动对焦成功才拍照，因此在模拟器上只能“预览”，无法真正拍照。

在真机上运行该程序，按下右下角的“拍照”键，将可以看到如图 11.9 所示的界面。

用户在图 11.9 所示对话框中输入照片的名称后，程序将会把拍得的照片保存到 SD 卡上。

运行该程序需要获得相机拍照的权限，因此需要在 AndroidManifest.xml 文件中增加如下代码片段进行授权：



图 11.9 拍照时自动对焦

```
<!-- 授予程序使用摄像头的权限 -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

注意： 该程序中第一行粗体字代码，控制该 SurfaceView 自己不需要自己维护缓冲区：sView.getHolder().setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);，按 Android 官方文档，该方法已经过时了，Android 系统会自动选择 SurfaceView 缓冲区的管理方式，但实际上在 Android 早期版本（比如 2.3.3）中依然需要调用该方法，否则该程序将会在 Camera 调用 startPreview()方法时引发异常。



11.3.2 录制视频短片

MediaRecorder 除了可用于录制音频之外，还可用于录制视频。使用 MediaRecorder 录制视频与录制音频的步骤基本相同。只是录制视频时不仅需要采集声音，还需要采集图像。为了让 MediaRecorder 录制时采集图像，应该在调用 setAudioSource(int audio_source)方法时再调用 setVideoSource(int video_source)方法来设置图像来源。

除此之外，还需在调用 setOutputFormat()设置输出文件格式之后进行如下步骤。

- 调用 MediaRecorder 对象的 setVideoEncoder()、setVideoEncodingBitRate(int bitRate)、setVideoFrameRate 设置所录制的视频的编码格式、编码位率、每秒多少帧等，这些参数将可以控制所录制的视频的品质、文件的大小。一般来说，视频品质越好，视频文件越大。
- 调用 MediaRecorder 的 setPreviewDisplay(Surface sv)方法设置使用哪个 SurfaceView 来显示视频预览。

剩下的代码则与录制音频的代码基本相同。

实例：录制生活短片

下面的实例示范了如何录制视频，该程序的界面中提供了两个按钮用于控制开始、结束录制；除此之外，程序界面中还提供了一个 SurfaceView 来显示视频预览，该程序的界面布局文件如下。

程序清单：codes\11\11.3\RecordVideo\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<!-- 显示视频预览的 SurfaceView -->
<SurfaceView
      android:id="@+id/sView"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
<LinearLayout
      android:orientation="horizontal"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center_horizontal"
      android:layout_alignParentBottom="true"
      android:layout_centerHorizontal="true">
      <ImageButton
            android:id="@+id/record"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/record" />
      <ImageButton
            android:id="@+id/stop"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/stop" />
</LinearLayout>
</RelativeLayout>
```

提供了上面所示的界面布局文件之后，接下来就可以在程序中使用 MediaRecorder 来录制视频了，录制视频与录制音频的步骤基本相似。只是需要额外设置视频的图像来源、视频格式等，除此之外还需要设置使用 SurfaceView 显示视频预览。录制视频的程序代码如下。

程序清单：codes\11\11.3\RecordVideo\src\org\crazyit\sound\RecordVideo.java

```
public class RecordVideo extends Activity
      implements OnClickListener
{
      // 程序中的两个按钮
      ImageButton record , stop;
      // 系统的视频文件
      File videoFile ;
      MediaRecorder mRecorder;
      // 显示视频预览的 SurfaceView
    SurfaceView sView;
    // 记录是否正在进行录制
    private boolean isRecording = false;
    @Override
```

```
public void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      // 获取程序界面中的两个按钮
      record = (ImageButton) findViewById(R.id.record);
      stop = (ImageButton) findViewById(R.id.stop);
      // 让 stop 按钮不可用
      stop.setEnabled(false);
      // 为两个按钮的单击事件绑定监听器
      record.setOnClickListener(this);
      stop.setOnClickListener(this);
      // 获取程序界面中的 SurfaceView
      sView = (SurfaceView) this.findViewById(R.id.sView);
  // 设置 Surface 不需要自己维护缓冲区
  sView.getHolder().setType(SurfaceHolder
      .SURFACE_TYPE_PUSH_BUFFERS);
  // 设置分辨率
  sView.getHolder().setFixedSize(320, 280);
  // 设置该组件让屏幕不会自动关闭
  sView.getHolder().setKeepScreenOn(true);
}
@Override
public void onClick(View source)
{
      switch (source.getId())
      {
            // 单击录制按钮
            case R.id.record:
                  if (!Environment.getExternalStorageState().equals(
                        android.os.Environment.MEDIA_MOUNTED))
                  {
                        Toast.makeText(RecordVideo.this
                              , "SD 卡不存在，请插入 SD 卡！"
                              , Toast.LENGTH_SHORT).show();
                        return;
                  }
                  try
                  {
                        // 创建保存录制视频的视频文件
                        videoFile = new File(Environment
                              .getExternalStorageDirectory()
                              .getCanonicalFile() + "/myvideo.mp4");
                        // 创建 MediaPlayer 对象
                        mRecorder = new MediaRecorder();
                        mRecorder.reset();
                        // 设置从麦克风采集声音
                        mRecorder.setAudioSource(MediaRecorder
                              .AudioSource.MIC);
                        // 设置从摄像头采集图像
                        mRecorder.setVideoSource(MediaRecorder
                              .VideoSource.CAMERA);
                        // 设置视频文件的输出格式
                        // 必须在设置声音编码格式、图像编码格式之前设置
                        mRecorder.setOutputFormat(MediaRecorder
                              .OutputFormat.MPEG_4);
                        // 设置声音编码的格式
                        mRecorder.setAudioEncoder(MediaRecorder
                              .AudioEncoder.DEFAULT);
                        // 设置图像编码的格式
                        mRecorder.setVideoEncoder(MediaRecorder
```

```
                        .VideoEncoder.MPEG_4_SP);
                    mRecorder.setVideoSize(320, 280);
                    // 每秒 4 帧
                    mRecorder.setVideoFrameRate(4);
                    mRecorder.setOutputFile(videoFile.getAbsolutePath());
                    // 指定使用 SurfaceView 来预览视频
                    mRecorder.setPreviewDisplay(sView
                        .getHolder().getSurface());  //①
                    mRecorder.prepare();
                    // 开始录制
                    mRecorder.start();
                    System.out.println("---recording---");
                    // 让 record 按钮不可用
                    record.setEnabled(false);
                    // 让 stop 按钮可用
                    stop.setEnabled(true);
                    isRecording = true;
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
                break;
            // 单击停止按钮
            case R.id.stop:
                // 如果正在进行录制
                if (isRecording)
                {
                    // 停止录制
                    mRecorder.stop();
                    // 释放资源
                    mRecorder.release();
                    mRecorder = null;
                    // 让 record 按钮可用
                    record.setEnabled(true);
                    // 让 stop 按钮不可用
                    stop.setEnabled(false);
                }
                break;
        }
    }
}
```

上面的程序中粗体字代码设置了视频所采集的图像来源，以及视频的压缩格式、视频分辨率等属性，程序的①号粗体字代码则用于设置使用指定 SurfaceView 显示指定视频预览。

运行该程序需要使用麦克风录制声音，需要使用摄像头采集图像，这些都需要授予相应的权限；不仅如此，由于该录制视频时视频文件增大得较快，可能需要使用外部存储器，因此需要对应用程序授予相应的权限。也就是需要在 AndroidManifest.xml 文件中增加如下授权配置：

```
<!-- 授予该程序录制声音的权限 -->
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<!-- 授予该程序使用摄像头的权限 -->
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 授予使用外部存储器的权限 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

当在模拟器上运行该程序时，由于模拟器上没有摄像头硬件支持，因此程序因此使用模

拟图像作为视频。在模拟器上运行该程序将看到如图 11.10 所示界面。

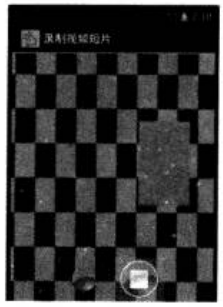


图 11.10 录制视频短片

11.4 本章小结

音频和视频都是非常重要的多媒体形式，Android 系统为音频、视频等多媒体的播放、录制提供了强大的支持。学习本章需要重点掌握如何使用 MediaPlayer、SoundPool 播放音频，如何使用 VideoView、MediaPlayer 播放视频。除此之外，还需要掌握通过 MediaRecorder 录制音频的方法，以及控制摄像头拍照、录制视频的方法。

CHAPTER

12

第12章 OpenGL与3D应用开发

本章要点

- 3D编程的概念和基本理论
- OpenGL与OpenGL ES
- Android的OpenGL ES支持
- 在Android环境中使用Open GL ES
- 利用OpenGL ES绘制2D图形
- 旋转图形
- 从2D到3D的转换
- 利用OpenGL ES绘制3D图形
- 通过OpenGL ES对3D图形应用贴图



前面介绍过 Android 系统的图形、图像处理。通过 Android 提供的图形、图形处理 API，开发者可以非常方便地处理二维图形的处理、开发 2D 游戏等。但现在这个时代的用户显然并不满足于 2D 操作界面和 2D 游戏，3D 技术已经被广泛应用于 PC 游戏上，3D 技术下一步将要占领的肯定是手机平台。而 Android 系统已经为 3D 技术准备好了，Android 系统完全内置了 OpenGL ES（OpenGL for Embedded System）支持，也就是说开发者可以在 Android 平台上使用 OpenGL ES API 来开发 3D 应用。

OpenGL 本身是高效、简洁的开放图形库接口，它定义了一个跨编程语言、跨平台的编程接口的规范，主要用于三维图形编程。但在手机等手持终端上运行 OpenGL 有些不太合适，所以 Android 系统内置的是 OpenGL ES 支持。本章将会向读者介绍 3D 开发的基本知识，并向读者介绍 OpenGL ES 编程的入门知识，但由于 OpenGL ES 的内容本身很多，而本书并不是一本专门介绍 OpenGL ES 编程的专著，所以并未过多地深入 OpenGL ES 编程，所以即使读者没有任何 3D 编程的知识，阅读本章也不会有任何障碍。

12.1 3D 图像与 3D 开发的基本知识

现在的互联网上，虽然还有一些 2D 游戏存在，但 3D 游戏已经逐渐成为主流。毕竟 3D 游戏具有更逼真的界面，能带给玩家更好的用户体验。就目前的技术来看，开发 3D 界面的各种技术已经非常成熟，为开发 3D 界面、3D 游戏提供了基础。

初学者可能会把 3D 开发想象得十分复杂。当然 3D 游戏开发肯定要比 2D 游戏开发更加复杂，毕竟开发 3D 界面需要的数据更多。

在介绍 3D 图形开发之前，我们先从 2D 图形开发入手。

先来看如何定义一个 2D 的三角形。对于 2D 的三角形来说，它只需要三个点，而且这三个点位于同一个平面上，因此程序只要为每个点指定 *X*、*Y* 两个坐标的值即可。

接下来看如何定义一个 3D 的三棱锥，一个三棱锥需要 4 个点，而且这 4 个点都不是位于同一个平面上，因此程序需要为每个点指定 *X*、*Y*、*Z* 三个坐标值。

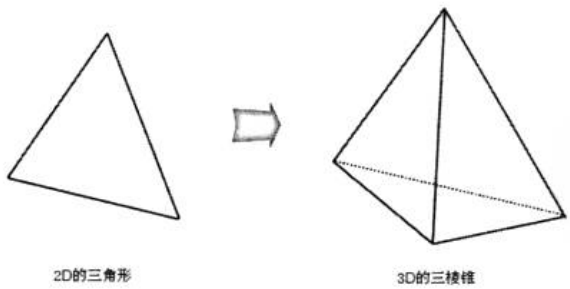


图 12.1 2D 到 3D 的转换

从图 12.1 可以看出，3D 图形需要处理的数据比 2D 图形要多得多。

图 12.1 看到的只是最简单的 2D 图形——三角形和最简单 3D 图形——三棱锥，但实际应用中，应用程序需要呈现出来的 2D 图形可能由很多曲线组成；类似地，应用程序需要呈现的 3D 图形也可能是由很多“曲面”组成的。

前面介绍 2D 绘图时，介绍了开发那个随手画图的程序，当程序希望在 2D 界面上绘制一条任意的曲线时，如果把这条曲线放大了来看（放得足够大），用户将会发现这个这条曲线

其实是由许多足够短的直线连接起来的。

对于一个 3D 图像来说，即使用户眼中看到的是一个“圆滑曲面”的 3D 图形，实际上它依然是由多个足够小的平面所组成的，图 12.2 就是从 3D MAX 里截取出来的一个 3D 图形。

正如图 12.2 所示：左边是用户希望看到的 3D 图形，但这个 3D 图形并不是程序员希望控制的。实际上程序员要控制的是右边的“网格图”，为了实现这个 3D 图形，程序员需要定义两方面的数据：

- 3D 图形的每个顶点（Vertex）的位置，每个顶点的位置都需要 *X*、*Y*、*Z* 三个坐标值。
- 3D 图形每个面由哪些顶点组成。

当程序给出了上面两方面的数据之后，接下来就可通知 3D 绘制接口来绘制这个 3D 图形了——就像需要绘制 2D 图形一样，程序员需要做的就是给出 2D 图形中各顶点的坐标。

为了定义 3D 图形每个顶点的位置，程序需要给出 *X*、*Y*、*Z* 三个坐标值。为此我们先要简单介绍一下 Android 的 3D 支持的三维坐标系统。

Android 的 3D 坐标系统与 2D 坐标系统完全不同，图 12.3 显示了 Android 的三维坐标系统。

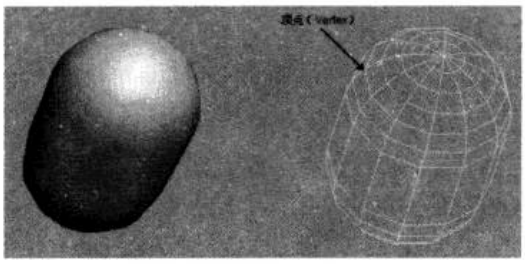


图 12.2　3D 图形的组成

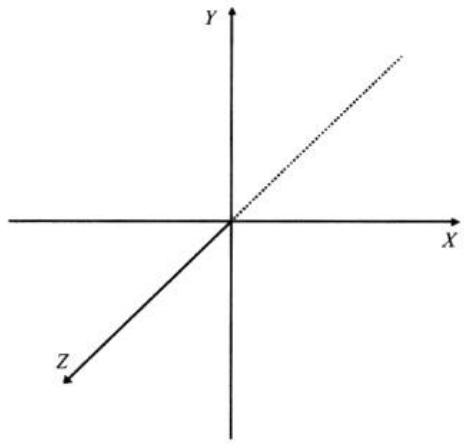


图 12.3　三维坐标系统

从图 12.3 可以看出，Android 的三维坐标系统中：坐标原点位于中央，*X* 轴从左向右延伸，原点左边的值为负数，右边为正数；*Y* 轴从下向上延伸，原点下边的值为负数，上边为正数；*Z* 轴屏幕里面向外面延伸，屏幕里面为负数，外面为正数。

了解了这个三维坐标系统，我们就可以根据该三维坐标系统来定义 3D 图形的每个顶点的位置了。

12.2　OpenGL 和 OpenGL ES 简介

OpenGL 的全称是 Open Graphics Library，即开放的图形库接口，它定义了一个跨编程语言、跨平台的编程接口的规范，它主要用于三维图形（实际上二维图形也可以）编程。OpenGL 的前身是 SGI 公司为其图形工作站开发的 IRIS GL。IRIS GL 是一个工业标准的 3D 图形软件接口，功能虽然强大但是移植性不好，于是 SGI 公司便在 IRIS GL 的基础上开发了 OpenGL。

OpenGL 体系简单，而且具有跨平台的特性，它不像 Direct3D（Microsoft 开发的 3D 图

形库接口，OpenGL 的最有力的竞争对手）只能在 Windows 系统上运行，因此 OpenGL 具有很广泛的适应性：它不仅适用于大型图形工作站，也适用于个人 PC。

在图形工作站、个人 PC 上，OpenGL 都可以工作良好，但三维图形计算必须需要处理大量数据，因此在一些如手机之类的小型设备上，如果希望使用 OpenGL 就比较困难了。为此，Khronos 集团为 OpenGL 提供了一个子集：OpenGL ES（OpenGL for Embedded System）。

Khronos 是一个图形软硬件行业协会，该协会主要关注图形和多媒体方面的开放标准，Khronos 协会针对手机、PDA 和游戏主机等嵌入式设备设置了 OpenGL ES。

OpenGL ES 是免费的、跨平台的、功能完善的 2D/3D 图形库接口 API，它针对多种嵌入式系统（包括控制台、移动电话、手持设备、家电设备和汽车）专门设计，它是一个精心提取出来的 OpenGL 的子集。

OpenGL ES 剔除了 OpenGl 中 glBegin/glEnd，四边形（GL_QUADS）、多边形（GL_POLYGONS）等许多非绝对必要的特性。经过多年发展，目前的 OpenGL ES 主要有两个版本，OpenGL ES 1.x 针对固定管线硬件；OpenGL ES 2.x 针对可编程管线硬件。

OpenGL ES 1.0 是以 OpenGL 1.3 规范为基础的，OpenGL ES 1.1 是以 OpenGL 1.5 规范为基础的，它们分别支持 common 和 common lite 两种 profile。lite profile 只支持定点实数，而 common profile 既支持定点数又支持浮点数，common profile 发布于 2005-8，引入了对可编程管线的支持。

目前 Android SDK 已经支持 OpenGL ES 2.0 的绝大部分功能，而且 Android 专门为 OpenGL 支持提供了 android.opengl 包，在该包下提供了 GLSurfaceView、GLU、GLUtils 等工具类，通过这些工具类在 Android 应用中使用 OpenGL ES 更加方便。

12.3 绘制 2D 图形

掌握了上面关于 3D 图形开发的基本知识之后，下面可以利用 Android 的 OpenGL ES 支持来进行 3D 开发了。在真正开发 3D 开发之前，还需要先学习 2D 开发。

➤➤12.3.1 在 Android 应用中使用 OpenGL ES

Android 为 OpenGL ES 支持提供了 GLSurfaceView 组件，这个组件用于显示 3D 图形。GLSurfaceView 本身并不提供绘制 3D 图形的功能，而是由 GLSurfaceView.Renderer 来完成了 SurfaceView 中 3D 图形的绘制。

归纳起来，在 Android 中使用 OpenGL ES 需要三个步骤。

① 创建 GLSurfaceView 组件，使用 Activity 来显示 GLSurfaceView 组件。

② 为 GLSurfaceView 组件创建 GLSurfaceView.Renderer 实例，实现 GLSurfaceView.Renderer 类时需要实现该接口里的三个方法。

- abstract void onDrawFrame(GL10 gl)：Renderer 对象调用该方法绘制 GLSurfaceView 的当前帧。
- abstract void onSurfaceChanged(GL10 gl, int width, int height)：当 GLSurfaceView 的大小改变时回调该方法。
- abstract void onSurfaceCreated(GL10 gl, EGLConfig config)：当 GLSurfaceView 被创建时回调该方法。

③ 调用 GLSurfaceView 组件的 setRenderer()方法指定 Renderer 对象，该 Renderer 对象将会完成 GLSurfaceView 里 3D 图形的绘制。

从上面的介绍不难看出，实际上绘制 3D 图形的难点不是如何使用 GLSurface 组件，而是如何实现 Renderer 类。实现 Renderer 类时需要实现三个方法。这三个方法都有一个 GL10 形参，它就代表了 GLOpen ES 的“绘制画笔”，读者可以把它想象成 Swing 2D 绘图中的 Graphics，也可以想象成 Android 2D 绘图中的 Canvas 组件——当我们希望 Renderer 绘制 3D 图形时，实际上是调用 GL10 的方法来进行绘制的。

当 SurfaceView 被创建时，系统会回调 Renderer 对象的 onSurfaceCreated()方法，该方法将可以对 OpenGL ES 执行一些无须任何改变的初始化，例如如下初始化代码：

```
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    // 关闭抗抖动
    gl.glDisable(GL10.GL_DITHER);
    // 设置系统对透视进行修正
    gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
    gl.glClearColor(0, 0 , 0 , 0);
    // 设置阴影平滑模式
    gl.glShadeModel(GL10.GL_SMOOTH);
    // 启用深度测试
    gl.glEnable(GL10.GL_DEPTH_TEST);
    // 设置深度测试的类型
    gl.glDepthFunc(GL10.GL_LEQUAL);
}
```

GL10 就是 OpenGL ES 的绘图接口，虽然这里看到的是一个 GL10，但实际上它也是 GL11 的实例，读者可通过（gl instanceof GL11）判断它是否为 GL11 接口的实例。

上面的方法中用到了 GL10 的一些初始化方法，关于这些方法的说明如下。

- glDisable(int cap)：该方法用于禁用 OpenGL ES 某个方面的特性。该方法中第一行代码用于关闭抗抖动，这样可以提高性能。
- glHint (int target, int mode)：该方法用于对 OpenGL ES 某方面进行修正。
- clearColor(float red, float green, float blue, float alpha)：该方法设置 OpenGL ES“清屏”所用的颜色，四个参数分别设置红、绿、蓝、透明度值：0 为最小值，1 为最大值。例如设置 gl.glClearColor(0, 0 , 0 , 0);就是用黑色“清屏”。
- glShadeModel(int mode)：该方法用于设置 OpenGL ES 的阴影模式。此处设为阴影平滑模式。
- glEnable (int cap)：该方法与 glDisable(int cap)方法相对，用于启用 OpenGL ES 某方面的特性，此处用于启动 OpenGL ES 的深度测试。所谓“深度测试”，就是让 OpenGL ES 负责跟踪每个物体在 *Z* 轴上的深度，这样就可避免后面的物体遮挡前面的物体。

当 SurfaceView 组件的大小发生改变时，系统会回调 Renderer 对象的 onSurfaceChanged()方法，因此该方法通常用于初始化 3D 场景。例如如下初始化代码：

```
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    // 设置 3D 视窗的大小及位置
    gl.glViewport(0, 0, width, height);
    // 将当前矩阵模式设为投影矩阵
```

```
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // 初始化单位矩阵
        gl.glLoadIdentity();
        // 计算透视视窗的宽度、高度比
        float ratio = (float)width/height;
        // 调用此方法设置透视视窗的空间大小
        gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    }
```

上面的方法中用到了GL10的一些初始化方法，关于这些方法的说明如下。

- glViewport(int x, int y, int width, int height)：设置3D视窗的位置与大小。其中前两个参数指定该视窗的位置；后两个参数指定该视窗的宽、高。
- glMatrixMode(int mode)：设置视图的矩阵模型。通常可接受GL10.GL_PROJECTION、GL10.GL_MODELVIEW两个常量值。
- 当调用glMatrixMode(GL10.GL_PROJECTION);代码后，指定将屏幕设为透视图（要想看到逼真的三维物体，这是必要的），这意味着越远的东西看起来越小；当调用glMatrixMode(GL10.GL_MODELVIEW);代码后，即将当前矩阵模式设为模型视图矩阵，这意味着任何新的变换都会影响该矩阵中的所有物体。
- glLoadIdentity()：相当于reset()方法，用于初始化单位矩阵。
- glFrustumf (float left, float right, float bottom, float top, float zNear, float zFar)：用于设置透视投影的空间大小。前两个参数用于设置*X*轴上的最小坐标值、最小坐标值；中间两个参数用于设置*Y*轴上的最小坐标值、最大坐标值；后两个参数用于设置*Z*轴上所能绘制的场景的深度的最小值、最大值。

例如我们调用如下代码：

```
gl.glFrustumf(-0.8, 0.8, -1, 1, 1, 10);
```

这意味着如果有一个二维矩形，它的四个顶点的坐标分别为：(-0.8, 1)、(0.8,1)、(0.8,-1)(-0.8,-1)，这个矩形将会占满整个视窗。



提示：

前面已经指出：三维坐标系统与二维坐标系统并不相同，二维坐标系统上的坐标值通常就直接使用系统的像素数量；但三维坐标系统的坐标值则取决于glFrustumf()方法的设置，当我们调用gl.glFrustumf(-0.8, 0.8, -1, 1, 1, 10);方法时，意味着该三维坐标系统的*X*轴的最左边的坐标值为-0.8，最右边的坐标值为0.8；*Y*轴最上面的坐标值为1.0，最下面的坐标值为-1.0。

GLSurfaceView上的所有3D图形都是由Renderer的onDrawFrame(GL10 gl)方法绘制出来的，重写该方法时就要把所有3D图形都绘制出来，该方法通常以如下形式开始：

```
@Override
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT|GL10.GL_DEPTH_BUFFER_BIT);
    ...
}
```

接下来在onDrawFrame方法中就可以调用GL10的方法开始绘制了。接下来会介绍GL10所提供的常见的绘制方法。

12.3.2 绘制平面上的多边形

前面已经说过，计算机里的3D图形其实是由许多个平面组合而成的。所谓“绘制3D图形”，其实是通过多个平面图形形成的，下面先从绘制平面图形开始。

调用GL10图形绘制2D图形的步骤如下：

① 调用GL10的glEnableClientState(GL10.GL_VERTEX_ARRAY);方法启用顶点坐标数组。

② 调用GL10的glEnableClientState(GL10.GL_COLOR_ARRAY);方法启用顶点颜色数组。

③ 调用GL10的glVertexPointer(int size, int type, int stride, Buffer pointer)方法设置顶点的位置数据。这个方法中pointer参数用于指定顶点坐标值，但这里并未使用三维数组来指定每个顶点 *X*、*Y*、*Z* 坐标的值，pointer 依然是一个一维数组，其格式为(x1,y1,z1,x2,y2,z2,x3,y3,z3...xN,yN,zN)；也就是该数组里将会包含3N个数值，每3个值指定一个顶点的 *X*、*Y*、*Z* 坐标值。第一个参数size指定多少个元素指定一个顶点位置，该size参数通常总是3；type参数指定顶点坐标值的类型，如果顶点坐标值为float类型，则指定为GL10.GL_FLOAT；如果顶点坐标值为整数，则指定为GL10.GL_FIXED。

④ 调用GL10的glColorPointer (int size, int type, int stride, Buffer pointer)方法设置顶点的颜色数据。这个方法中pointer参数用于指定顶点的颜色值，pointer依然是一个一维数组，其格式为(r1,g1,b1,a1,x2,y2,z2,a2,x3,y3,z3,a3...xN,yN,zN,aN)；也就是该数组里将会包含4*N*个数值，每4个值指定一个顶点的红、绿、蓝、透明度的颜色值。第一个参数size指定多少个元素指定一个顶点位置，该size参数通常总是4；type参数指定顶点坐标值的类型，如果顶点坐标值为float类型，则指定为GL10.GL_FLOAT；如果顶点坐标值为整数，则指定为GL10.GL_FIXED。

⑤ 调用GL10的glDrawArrays (int mode, int first, int count)方法绘制平面。该方法的第一个参数用于指定绘制图形类型，第二个参数指定从哪个顶点开始绘制，第三个参数指定总共绘制的顶点数量。

⑥ 绘制完成后，调用GL10的glFinish()方法结束绘制；并调用glDisableClientState (int)方法来停用顶点坐标数据、顶点颜色数据。

掌握上面的步骤之后，接下来通过示例程序来绘制几个简单的图形。

先为该程序提供一个Renderer实现类，该实现类的代码如下。

程序清单：codes\12\12.3\polygon\src\org\crazyit\opengl\MyRenderer.java

```
public class MyRenderer implements Renderer
{
    float[] triangleData = new float[] {
        0.1f, 0.6f , 0.0f , // 上顶点
        -0.3f, 0.0f , 0.0f , // 左顶点
        0.3f, 0.1f , 0.0f  // 右顶点
    };
    int[] triangleColor = new int[] {
        65535, 0, 0, 0, // 上顶点红色
        0, 65535, 0, 0, // 左顶点绿色
        0, 0, 65535, 0 // 右顶点蓝色
    };
    float[] rectData = new float[] {
        0.4f, 0.4f , 0.0f, // 右上顶点
```

```
        0.4f, -0.4f , 0.0f, // 右下顶点
        -0.4f, 0.4f , 0.0f, // 左上顶点
        -0.4f, -0.4f , 0.0f // 左下顶点
};
int[] rectColor = new int[] {
        0, 65535, 0, 0, // 右上顶点绿色
        0, 0, 65535, 0, // 右下顶点蓝色
        65535, 0, 0, 0, // 左上顶点红色
        65535, 65535, 0, 0 // 左下顶点黄色
};
// 依然是正方形的4个顶点，只是顺序交换了一下
float[] rectData2 = new float[] {
        -0.4f, 0.4f , 0.0f, // 左上顶点
        0.4f, 0.4f , 0.0f, // 右上顶点
        0.4f, -0.4f , 0.0f, // 右下顶点
        -0.4f, -0.4f , 0.0f // 左下顶点
};
float[] pentacle = new float[]{
        0.4f , 0.4f , 0.0f,
        -0.2f , 0.3f , 0.0f,
        0.5f , 0.0f , 0f,
        -0.4f , 0.0f , 0f,
        -0.1f, -0.3f , 0f
};
FloatBuffer triangleDataBuffer;
IntBuffer triangleColorBuffer;
FloatBuffer rectDataBuffer;
IntBuffer rectColorBuffer;
FloatBuffer rectDataBuffer2;
FloatBuffer pentacleBuffer;
public MyRenderer()
{
        // 将顶点位置数据数组转换成FloatBuffer
        triangleDataBuffer = floatBufferUtil(triangleData);
        rectDataBuffer = floatBufferUtil(rectData);
        rectDataBuffer2 = floatBufferUtil(rectData2);
        pentacleBuffer = floatBufferUtil(pentacle);
        // 将顶点颜色数据数组转换成IntBuffer
        triangleColorBuffer = intBufferUtil(triangleColor);
        rectColorBuffer = intBufferUtil(rectColor);
}
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
        // 关闭抗抖动
        gl.glDisable(GL10.GL_DITHER);
        // 设置系统对透视进行修正
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT
                , GL10.GL_FASTEST);
        gl.glClearColor(0, 0, 0, 0);
        // 设置阴影平滑模式
        gl.glShadeModel(GL10.GL_SMOOTH);
        // 启用深度测试
        gl.glEnable(GL10.GL_DEPTH_TEST);
        // 设置深度测试的类型
        gl.glDepthFunc(GL10.GL_LEQUAL);
}
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
        // 设置3D视窗的大小及位置
```

```
        gl.glViewport(0, 0, width, height);
        // 将当前矩阵模式设为投影矩阵
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // 初始化单位矩阵
        gl.glLoadIdentity();
        // 计算透视视窗的宽度、高度比
        float ratio = (float) width / height;
        // 调用此方法设置透视视窗的空间大小
        gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
    }
    // 绘制图形的方法
    @Override
    public void onDrawFrame(GL10 gl)
    {
        // 清除屏幕缓存和深度缓存
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        // 启用顶点坐标数据
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // 启用顶点颜色数据
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        // 设置当前矩阵堆栈为模型堆栈
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        // ---------------------绘制第一个图形---------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(-0.32f, 0.35f, -1f);  //①
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleDataBuffer);
        // 设置顶点的颜色数据
        gl.glColorPointer(4, GL10.GL_FIXED, 0, triangleColorBuffer);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
        // ---------------------绘制第二个图形---------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(0.6f, 0.8f, -1.5f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer);
        // 设置顶点的颜色数据
        gl.glColorPointer(4, GL10.GL_FIXED, 0, rectColorBuffer);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        // ---------------------绘制第三个图形---------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(-0.4f, -0.5f, -1.5f);
        // 设置顶点的位置数据（依然使用之前的顶点颜色）
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer2);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        // ---------------------绘制第四个图形---------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(0.4f, -0.5f, -1.5f);
        // 设置使用纯色填充
        gl.glColor4f(1.0f, 0.2f, 0.2f, 0.0f);  //②
        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, pentacleBuffer);
        // 根据顶点数据绘制平面图形
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 5);
```

```
            // 绘制结束
            gl.glFinish();
            gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        }
        // 定义一个工具方法，将 int[]数组转换为 OpenGL ES 所需的 IntBuffer
        private IntBuffer intBufferUtil(int[] arr)
        {
            IntBuffer mBuffer;
            // 初始化 ByteBuffer，长度为 arr 数组的长度*4，因为一个 int 占 4 个字节
            ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
            // 数组排列用 nativeOrder
            qbb.order(ByteOrder.nativeOrder());
            mBuffer = qbb.asIntBuffer();
            mBuffer.put(arr);
            mBuffer.position(0);
            return mBuffer;
        }
        // 定义一个工具方法，将 float[]数组转换为 OpenGL ES 所需的 FloatBuffer
        private FloatBuffer floatBufferUtil(float[] arr)
        {
            FloatBuffer mBuffer;
            // 初始化 ByteBuffer，长度为 arr 数组的长度*4，因为一个 int 占 4 个字节
            ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
            // 数组排列用 nativeOrder
            qbb.order(ByteOrder.nativeOrder());
            mBuffer = qbb.asFloatBuffer();
            mBuffer.put(arr);
            mBuffer.position(0);
            return mBuffer;
        }
    }
```

在本书的上一版中，为了将 int[]数组转换为 Open GL ES 所需的 IntBuffer，只要调用 IntBuffer 的 wrap()方法进行包装即可，但现在 Android 平台的要求更加严格，要求该 Buffer 必须是 native Buffer（因此使用 ByteBuffer 的 allocateDirect()方法进行创建），并且该 Buffer 必须是排序的（因此调用了 BufferBuffer 的 order()方法进行排序）。

上面的程序中粗体字代码就是使用 GL10 绘制图形的关键代码：加载顶点位置数据；加载顶点颜色数据；调用 GL10 的 glDrawArrays()方法绘制即可。由于加载顶点数据、顶点颜色数据时都需要 Buffer 对象，因此程序在 MyRenderer 类的构造器中把这些顶点数据、顶点颜色数据都包装成了相应的 FloatBuffer、IntBuffer。

在上面的程序中①号代码调用了 GL10 的 glTranslatef(-0.32f, 0.35f, -1f)方法，这个 glTranslatef()方法的作用就类似于 Android 2D 绘图中 Matrix 的 setTranslate(float dx, float dy)方法，它们都用于移动绘图中心，区别只是 2D 绘图中 Matrix 的 setTranslate()方法只要指定在 *X*、*Y* 轴上的移动距离，而 GL10 的 glTranslatef()需要指定在 *X*、*Y*、*Z* 轴向上移动距离。在绘制图形之前，先调用 GL10 的 glTranslatef(float,float,float)方法即可保证把图形绘制在指定的中心点。

上面的程序中②号代码还调用了 glColor4f(1.0f, 0.2f, 0.2f, 0.0f);方法设置使用纯色填充，设置使用纯色填充时需要禁用顶点颜色数组，如②号代码后的一行代码。

在 Activity 中定义一个 GLSurfaceView，并使用上面的 Renderer 进行绘制，程序如下。

程序清单：codes\12\12.3\polygon\src\org\crazyit\opengl\Polygon.java

```
public class Polygon extends Activity
{
    @Override
```

```
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            // 创建一个 GLSurfaceView，用于显示 OpenGL 绘制的图形
            GLSurfaceView glView = new GLSurfaceView(this);
            // 创建 GLSurfaceView 的内容绘制器
            MyRenderer myRender = new MyRenderer();
            // 为 GLSurfaceView 设置绘制器
            glView.setRenderer(myRender);
            setContentView(glView);
        }
}
```

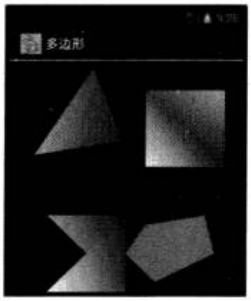


图 12.4　使用 OpenGL ES 绘制 2D 图形

运行上面的程序，看到如图 12.4 所示的输出。

很多读者会对图 12.4 所示绘制的图形感到奇怪：第二个图形（右上角的正方形）和第三个图形（左下角的图形）都有完全相同的四个坐标点，只是定义 4 个坐标点的顺序略有不同，为何绘制的图形存在这么大区别呢？

再来看 GL10 提供的 glDrawArrays(int mode, int first, int count) 方法，该方法第一个参数指定绘制的模式，可指定为如下两个值。

- GL10.GL_TRIANGLES：绘制三角形。
- GL10.GL_TRIANGLE_STRIP：用多个三角形来绘制多边形。



提示：

前面介绍 OpenGL 与 OpenGL ES 的区别时已经指出：OpenGL ES 剔除了 OpenGl 中的四边形（GL_QUADS）、多边形（GL_POLYGONS）支持，也就是 OpenGL ES 只能绘制三角形组成 3D 图形。

当调用 glDrawArrays 方法时，如果将 mode 参数指定为 GL10.GL_TRIANGLES，就绘制简单的三角形；如果将 mode 参数指定为 GL10.GL_TRIANGLE_STRIP，系统将会沿着给出的顶点数据来绘制三角形。

对于上面程序中的第二个图形，程序给出 4 个顶点的顺序，如图 12.5 所示。

当指定了 glDrawArrays(int mode, int first, int count) 方法的第一个参数为 GL10.GL_TRIANGLE_STRIP 时，系统总会从 first 个顶点开始，每 3 个顶点绘制一个三角形。例如调用如下代码：gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);，这意味着将会绘制两个三角形，分别由 0、1、2 三个顶点组成的三角形、1、2、3 三个顶点所组成的三角形。因此对于第二个图形而言，gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);所绘制的图形如图 12.6 所示。

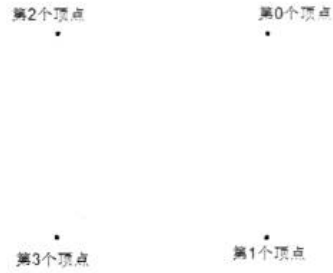


图 12.5　第二个图形的顶点数据

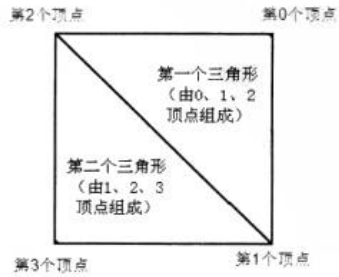


图 12.6　OpenGL ES 使用三角形绘制正方形

对于第三个图形而言，虽然它的 4 个顶点的位置与第二个顶点的位置完全相同，但由于 4 个顶点的顺序有所不同，所以它也是绘制两个三角形（由 0、1、2 顶点组成第一个三角形、1、2、3 顶点组成第二个三角形），如图 12.7 所示（浅色边界为第二个三角形的边界）。

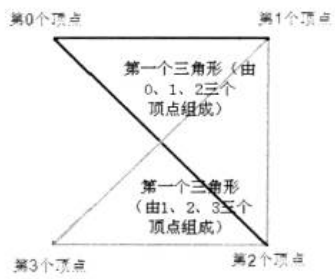


图 12.7 OpenGL ES 使用三角形绘制图形

如图 12.7 所示的图形就与前面上面程序的所绘制的第三个图形相同了。

讲到这里，可能有读者对 3D 开发感到害怕了：现在还只是绘制了几个简单的图形，程序员不仅要给出图形每个顶点的位置信息，还要按指定顺序来排列这些顶点，这也太复杂了吧！不要担心，现在所使用的 glDrawArrays()方法是有点复杂，实际上 3D 图形中每个顶点的坐标值不需要由程序员计算、给出；顶点的排列顺序也无须由程序员排列。



提示：

可以想象，如果 3D 场景中每个物体的所有顶点都由程序员来计算、定义，那几乎是不可想象的——如果让我们定义一个怪兽的头部，需要多少个顶点？每个顶点的位置到底应该在哪里？把一个程序员算到死都有可能！这时候通常会借助于 3D MAX、Maya 等三维建模工具，当我们把一个怪兽头部的模型建立出来后，这个物体的所有顶点坐标值及顶点的排列顺序都可以导出来。OpenGL 可以直接导入这些三维建模工具所建立的模型。

12.3.3 旋转

GL10 提供了一个 glRotatef(float angle, float x, float y, float z)方法，该方法用于控制旋转，该方法中 angle 控制旋转角度；而 x、y、z 参数则共同决定了旋转轴的方向。

本质上，glRotatef(float angle, float x, float y, float z)方法的作用与 glTranslatef(float x, float y, float z)方法相似，只是 glTranslatef(float x, float y, float z)方法控制图形中心移动；而 glRotatef(float angle, float x, float y, float z)方法控制图形沿着指定旋转轴转动指定角度。

因此只要在调用 glTranslatef()方法控制图形移动之后，再调用 glRotatef()控制图形旋转即可，如果希望看到指定图形不断旋转，只要在 onDrawFrame(GL10 gl)方法中不断增加旋转角度即可。

下面是该程序所用的 Renderer 实现类。

程序清单：codes\12\12.3\rotatePolygon\src\org\crazyit\opengl\MyRenderer.java

```
public class MyRenderer implements Renderer
{
    // 省略定义顶点坐标的代码
    ...
    FloatBuffer triangleDataBuffer;
    IntBuffer triangleColorBuffer;
    FloatBuffer rectDataBuffer;
    IntBuffer rectColorBuffer;
    FloatBuffer rectDataBuffer2;
    FloatBuffer pentacleBuffer;
    // 控制旋转的角度
    private float rotate;
```

```
public MyRenderer()
{
    // 将顶点位置数据数组包装成 FloatBuffer
    triangleDataBuffer = floatBufferUtil(triangleData);
    rectDataBuffer = floatBufferUtil(rectData);
    rectDataBuffer2 = floatBufferUtil(rectData2);
    pentacleBuffer = floatBufferUtil(pentacle);
    // 将顶点颜色数据数组包装成 IntBuffer
    triangleColorBuffer = intBufferUtil(triangleColor);
    rectColorBuffer = intBufferUtil(rectColor);
}
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
    //省略该方法的代码
    ...
}
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
    // 省略该方法的代码
    ...
}
// 绘制图形的方法
@Override
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // 启用顶点坐标数据
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    // 启用顶点颜色数据
    gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
    // 设置当前矩阵堆栈为模型堆栈
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    // --------------------绘制第一个图形---------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(-0.32f, 0.35f, -1f);
    // 设置顶点的位置数据
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, triangleDataBuffer);
    // 设置顶点的颜色数据
    gl.glColorPointer(4, GL10.GL_FIXED, 0, triangleColorBuffer);

    // 根据顶点数据绘制平面图形
    gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    // --------------------绘制第二个图形---------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(0.6f, 0.8f, -1.5f);
    gl.glRotatef(rotate, 0f, 0f, 0.1f);
    // 设置顶点的位置数据
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer);
    // 设置顶点的颜色数据
    gl.glColorPointer(4, GL10.GL_FIXED, 0, rectColorBuffer);
    // 根据顶点数据绘制平面图形
    gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
    // --------------------绘制第三个图形---------------------
    // 重置当前的模型视图矩阵
    gl.glLoadIdentity();
    gl.glTranslatef(-0.4f, -0.5f, -1.5f);
```

```
            gl.glRotatef(rotate, 0f, 0.2f, 0f);
            // 设置顶点的位置数据（依然使用之前的顶点颜色）
            gl.glVertexPointer(3, GL10.GL_FLOAT, 0, rectDataBuffer2);
            //根据顶点数据绘制平面图形
            gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
            // --------------------绘制第四个图形--------------------
            // 重置当前的模型视图矩阵
            gl.glLoadIdentity();
            gl.glTranslatef(0.4f, -0.5f, -1.5f);
            // 设置使用纯色填充
            gl.glColor4f(1.0f, 0.2f, 0.2f, 0.0f);
            gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
            // 设置顶点的位置数据
            gl.glVertexPointer(3, GL10.GL_FLOAT, 0, pentacleBuffer);
            // 根据顶点数据绘制平面图形
            gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 5);
            // 绘制结束
            gl.glFinish();
            gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
            // 旋转角度增加 1
            rotate+=1;
        }
        // 省略 intBufferUtil 和 floatBufferUtil 两个工具方法的代码
        ...
    }
```

上面的程序中粗体字代码定义了一个 rotate 变量，该变量用于控制程序中两个图形的旋转角度，其中第二个图形沿着 *Z* 轴旋转，第三个图形则沿着 *Y* 轴旋转。运行该程序，将可以看到第二个图形、第三个图形不断旋转的效果，如图 12.8 所示。

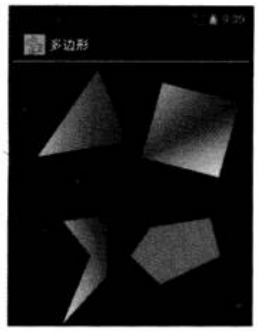


图 12.8 控制图形转换

掌握了使用 OpenGL ES 通过三角形绘制平面图形之后，接下来就可以调用 OpenGL ES 来绘制 3D 图形了——绘制 3D 图形需要定义更多顶点数据，而且需要更好地控制哪些顶点需要组成三角形。

12.4 绘制 3D 图形

如果定义的顶点不在同一个平面上，并且使用三角形把合适的顶点连接起来，就可以绘制出 3D 图形了。

12.4.1 构建 3D 图形

正如前面介绍的，使用 OpenGL ES 绘制 3D 图形的方法与绘制 2D 图形的步骤大致相同，只是绘制 3D 图形需要定义更多的顶点数据，而且 3D 图形需要绘制更多的三角形。

如果还使用前面介绍的 glDrawArrays(int mode, int first, int count)进行绘制，我们将会面临一个巨大的考验：到底要按怎样的顺序来组织三维空间上的多个顶点，才能绘制出我们需要的三角形？为了解决这个问题，GL10 提供了如下方法。

- glDrawElements (int mode, int count, int type, Buffer indices)：根据 indices 指定的索引点来绘制三角形。该方法的第一个参数指定绘制的图形的类型，可设为 GL10.GL_TRIANGLES 或 GL10.GL_TRIANGLE_STRIP；第二个参数指定一共包含多少个顶点。indices 参数最重要，它包装了一个长度为 3N 的数组，比如让该参

数包装{0,2,3,1,4,5}数组，这意味着告诉 OpenGL ES 要绘制两个三角形，第一个三角形的三个顶点为 0、2、3 个顶点，第二个三角形的三个顶点为 1、4、5 个顶点。

由此可见，如果希望在程序中使用 glDrawElements (int mode, int count, int type, Buffer indices)方法来绘制 3D 图形，不仅需要指定每个 3D 图形的顶点位置信息，也需要指定 3D 图形的每个面由哪三个顶点组成。

下面的程序使用 glDrawElements (int mode, int count, int type, Buffer indices)方法绘制了两个简单的 3D 图形：三棱锥和立方体，该程序的 Activity 依然没有变化。故下面只给出 Renderer 实现类的代码。

程序清单：codes\12\12.4\Simple3D\src\org\crazyit\opengl\MyRenderer.java

```
public class MyRenderer implements Renderer
{
    // 定义三棱椎的 4 个顶点
    float[] taperVertices = new float[] {
        0.0f, 0.5f, 0.0f,
        -0.5f, -0.5f, -0.2f,
        0.5f, -0.5f, -0.2f,
        0.0f, -0.2f, 0.2f
    };
    // 定义三棱椎的 4 个顶点的颜色
    int[] taperColors = new int[]{
        65535, 0, 0, 0,  // 红色
        0, 65535, 0, 0,  // 绿色
        0, 0, 65535, 0,  // 蓝色
        65535, 65535, 0, 0 //黄色
    };
    // 定义三棱椎的 4 个三角面
    private byte[] taperFacets = new byte[]{
        0, 1, 2, //0、1、2 三个顶点组成一个面
        0, 1, 3, //0、1、3 三个顶点组成一个面
        1, 2, 3, //1、2、3 三个顶点组成一个面
        0, 2, 3 //0、2、3 三个顶点组成一个面
    };
    // 定义立方体的 8 个顶点
    float[] cubeVertices = new float[] {
        // 上顶面正方形的 4 个顶点
        0.5f, 0.5f, 0.5f,
        0.5f, -0.5f, 0.5f,
        -0.5f, -0.5f, 0.5f,
        -0.5f, 0.5f, 0.5f,
        // 下底面正方形的 4 个顶点
        0.5f, 0.5f, -0.5f,
        0.5f, -0.5f, -0.5f,
        -0.5f, -0.5f, -0.5f,
        -0.5f, 0.5f, -0.5f
    };
    // 定义立方体所需要的 6 个面（一共是 12 个三角形所需的顶点）
    private byte[] cubeFacets = new byte[]{
        0, 1, 2,
        0, 2, 3,
        2, 3, 7,
        2, 6, 7,
        0, 3, 7,
        0, 4, 7,
        4, 5, 6,
        4, 6, 7,
        0, 1, 4,
```

```
        1, 4, 5,
        1, 2, 6,
        1, 5, 6
};
// 定义Open GL ES绘制所需要的Buffer对象
FloatBuffer taperVerticesBuffer;
IntBuffer taperColorsBuffer;
ByteBuffer taperFacetsBuffer;
FloatBuffer cubeVerticesBuffer;
ByteBuffer cubeFacetsBuffer;
// 控制旋转的角度
private float rotate;
public MyRenderer()
{
        // 将三棱椎的顶点位置数据数组包装成FloatBuffer
        taperVerticesBuffer = floatBufferUtil(taperVertices);
        // 将三棱椎的4个面的数组包装成ByteBuffer
        taperFacetsBuffer = ByteBuffer.wrap(taperFacets);
        // 将三棱椎的4个定点的颜色数组包装成IntBuffer
        taperColorsBuffer = intBufferUtil(taperColors);
        // 将立方体的顶点位置数据数组包装成FloatBuffer
        cubeVerticesBuffer = floatBufferUtil(cubeVertices);
        // 将立方体的6个面（12个三角形）的数组包装成ByteBuffer
        cubeFacetsBuffer = ByteBuffer.wrap(cubeFacets);
}
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config)
{
        // 关闭抗抖动
        gl.glDisable(GL10.GL_DITHER);
        // 设置系统对透视进行修正
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_FASTEST);
        gl.glClearColor(0, 0, 0, 0);
        // 设置阴影平滑模式
        gl.glShadeModel(GL10.GL_SMOOTH);
        // 启用深度测试
        gl.glEnable(GL10.GL_DEPTH_TEST);
        // 设置深度测试的类型
        gl.glDepthFunc(GL10.GL_LEQUAL);
}
@Override
public void onSurfaceChanged(GL10 gl, int width, int height)
{
        // 设置3D视窗的大小及位置
        gl.glViewport(0, 0, width, height);
        // 将当前矩阵模式设为投影矩阵
        gl.glMatrixMode(GL10.GL_PROJECTION);
        // 初始化单位矩阵
        gl.glLoadIdentity();
        // 计算透视视窗的宽度、高度比
        float ratio = (float) width / height;
        // 调用此方法设置透视视窗的空间大小。
        gl.glFrustumf(-ratio, ratio, -1, 1, 1, 10);
}
// 绘制图形的方法
@Override
public void onDrawFrame(GL10 gl)
{
        // 清除屏幕缓存和深度缓存
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        // 启用顶点坐标数据
```

```
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        // 启用顶点颜色数据
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        // 设置当前矩阵模式为模型视图。
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        // --------------------绘制第一个图形---------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(-0.6f, 0.0f, -1.5f);
        // 沿着 Y 轴旋转
        gl.glRotatef(rotate, 0f, 0.2f, 0f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, taperVerticesBuffer);
        // 设置顶点的颜色数据
        gl.glColorPointer(4, GL10.GL_FIXED, 0, taperColorsBuffer);
        // 按 taperFacetsBuffer 指定的面绘制三角形
        gl.glDrawElements(GL10.GL_TRIANGLE_STRIP
            , taperFacetsBuffer.remaining(),
            GL10.GL_UNSIGNED_BYTE, taperFacetsBuffer);
        // --------------------绘制第二个图形---------------------
        // 重置当前的模型视图矩阵
        gl.glLoadIdentity();
        gl.glTranslatef(0.7f, 0.0f, -2.2f);
        // 沿着 Y 轴旋转
        gl.glRotatef(rotate, 0f, 0.2f, 0f);
        // 沿着 X 轴旋转
        gl.glRotatef(rotate, 1f, 0f, 0f);
        // 设置顶点的位置数据
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, cubeVerticesBuffer);
        // 不设置顶点的颜色数据，还用以前的颜色数据
        // 按 cubeFacetsBuffer 指定的面绘制三角形
        gl.glDrawElements(GL10.GL_TRIANGLE_STRIP
            , cubeFacetsBuffer.remaining(),
            GL10.GL_UNSIGNED_BYTE, cubeFacetsBuffer);
        // 绘制结束
        gl.glFinish();
        gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
        // 旋转角度增加 1
        rotate+=1;
    }
    // 定义一个工具方法，将 int[]数组转换为 OpenGL ES 所需的 IntBuffer
    private IntBuffer intBufferUtil(int[] arr)
    {
        IntBuffer mBuffer;
        // 初始化 ByteBuffer，长度为 arr 数组的长度*4，因为一个 int 占 4 个字节
        ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
        // 数组排列用 nativeOrder
        qbb.order(ByteOrder.nativeOrder());
        mBuffer = qbb.asIntBuffer();
        mBuffer.put(arr);
        mBuffer.position(0);
        return mBuffer;
    }
    // 定义一个工具方法，将 float[]数组转换为 OpenGL ES 所需的 FloatBuffer
    private FloatBuffer floatBufferUtil(float[] arr)
    {
        FloatBuffer mBuffer;
        // 初始化 ByteBuffer，长度为 arr 数组的长度*4，因为一个 int 占 4 个字节
        ByteBuffer qbb = ByteBuffer.allocateDirect(arr.length * 4);
        // 数组排列用 nativeOrder
        qbb.order(ByteOrder.nativeOrder());
```

```
            mBuffer = qbb.asFloatBuffer();
            mBuffer.put(arr);
            mBuffer.position(0);
            return mBuffer;
        }
    }
```

从上面的程序不难看出，绘制 3D 图形的步骤与绘制 2D 图形的步骤基本相似，区别只是绘制 3D 图形不仅需要定义各顶点位置的坐标，还需要定义 3D 图形的各个三角面由哪些顶点组成，例如上面的程序中粗体字代码所示：为了定义一个立方体，除了给出这个立方体的 8 个顶点位置坐标之外，还需要定义组成该立方体的 12 个三角形。如图 12.9 显示了该立方体上 8 个顶点的位置。

接下来程序需要把图 12.9 所示正方形的 6 个面（由 12 个三角形组成）定义出来，依次指定每个三角形包括哪三个顶点，也就是使程序中粗体字代码所定义的数组。

运行上面的程序，将可以看到如图 12.10 所示的 3D 图形。

从上面的程序可以看出，为了在程序中绘制一个简单的 3D 立方体，程序需要定义 12 个三角面，这种规则的立方体还可以由程序员计算并指定，但如果遇到更复杂的 3D 物体，就必须借助于三维建模软件了。

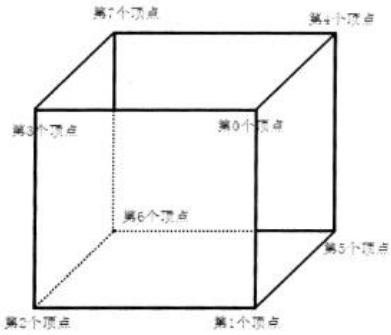


图 12.9 立方体的 8 个顶点的位置示意

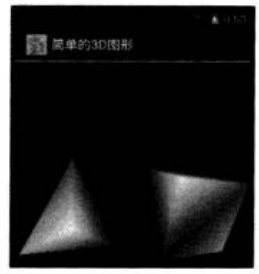


图 12.10 OpenGL 绘制 3D 图形

12.4.2 应用纹理贴图

为了让 3D 图形更加逼真，我们需要为这些 3D 图形应用纹理贴图，比如开发一个怪兽头部，如果只是为这个头部绘制五颜六色的“皮肤”，那也太假了吧。如果考虑为这个怪兽应用“鳄鱼皮肤”的纹理图，或者应用“蟒蛇皮肤”的纹理贴图，这个怪兽头部就“逼真”得多了，甚至会带给人一种恐怖的感觉了。

为了在 OpenGL ES 中启用纹理贴图功能，可以在 Renderer 实现类的 onSurfaceCreated (GL10 gl, EGLConfig config)方法中启动纹理贴图，例如如下代码：

```
// 启用 2D 纹理贴图
gl.glEnable(GL10.GL_TEXTURE_2D);
```

接下来就需要准备一张图片来作为纹理贴图了，建议该图片的长宽是 2 的 N 次方，比如长、宽为 256，512 等。把这张准备贴图的位图放在 Android 项目的 res/drawable-mdpi 目录下，方便应用程序加载该图片资源。

接下来程序开始记载该图片并生成对应的纹理贴图，例如如下方法。

```
// 加载位图
```

```
bitmap = BitmapFactory.decodeResource(context.getResources(),
      R.drawable.sand);
int[] textures = new int[1];
// 指定生成N个纹理（第一个参数指定生成一个纹理）
// textures 数组将负责存储所有纹理的代号
gl.glGenTextures(1, textures, 0);
// 获取 textures 纹理数组中的第一个纹理
texture = textures[0];
// 通知 OpenGL 将 texture 纹理绑定到 GL10.GL_TEXTURE_2D 目标中
gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
// 设置纹理被缩小（距离视点很远时被缩小）时候的滤波方式
gl.glTexParameterf(GL10.GL_TEXTURE_2D,
      GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
// 设置纹理被放大（距离视点很近时被方法）时候的滤波方式
gl.glTexParameterf(GL10.GL_TEXTURE_2D,
      GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
// 设置在横向、纵向上都是平铺纹理
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
      GL10.GL_REPEAT);
gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
      GL10.GL_REPEAT);
// 加载位图生成纹理
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
```

上面的程序中用到了 GL10 的如下方法。

- glGenTextures(int n, int[] textures, int offset)：该方法指定一次性生成 n 个纹理，该方法所生成的纹理的代号放入其中 textures 数组中，offset 指定从第几个数组元素开始存放纹理代号。
- glBindTexture(int target, int texture)：该方法用于将 texture 纹理绑定到 target 目标上。
- glTexParameterf(int target, int pname, float param)：该方法用于为 target 纹理目标设置属性，其中第二个参数是属性名，第三个参数是属性值。

上面的程序中有 4 行代码调用了 glTexParameterf(int target, int pname, float param)方法，程序设置了当纹理被放大时使用 GL10.GL_LINEAR 滤波方式；当纹理被缩小时使用 GL10.GL_NEAREST 滤波方式；一般来说，使用 GL10.GL_LINEAR 滤波方式有较好的效果，但系统开销略微大了一些，具体采用哪种滤波方式则取决于目标机器本身的性能。

上面代码的最后一行调用了 GLUtils 工具类的方法来加载指定位图，并根据位图来生成纹理，通过上面的代码即可得到一个用于贴图的纹理了。

在 3D 绘制中进行纹理贴图也很简单，与设置顶点颜色的步骤相似，只要三步：

1. 设置启用贴图坐标数组。
2. 设置贴图坐标的数组信息。
3. 调用 GL10 的 glBindTexture (int target, int texture)方法执行贴图。

下面的程序示范如何为一个立方体进行贴图，而且这个程序提供了手势检测器，允许用户通过手势来改变该立方体的角度。程序代码如下。

程序清单：codes\12\12.4\Texture3D\src\org\crazyit\opengl\Texture3D.java

```
public class Texture3D extends Activity
      implements OnGestureListener
{
      // 定义旋转角度
      private float anglex = 0f;
      private float angley = 0f;
```

```
static final float ROTATE_FACTOR = 60;
// 定义手势检测器实例
GestureDetector detector;
@Override
public void onCreate(Bundle savedInstanceState)
{
      super.onCreate(savedInstanceState);
      // 创建一个 GLSurfaceView，用于显示 OpenGL 绘制的图形
      GLSurfaceView glView = new GLSurfaceView(this);
      // 创建 GLSurfaceView 的内容绘制器
      MyRenderer myRender = new MyRenderer(this);
      // 为 GLSurfaceView 设置绘制器
      glView.setRenderer(myRender);
      setContentView(glView);
      // 创建手势检测器
      detector = new GestureDetector(this);
}
@Override
public boolean onTouchEvent(MotionEvent me)
{
      // 将该 Activity 上的触碰事件交给 GestureDetector 处理
      return detector.onTouchEvent(me);
}
@Override
public boolean onFling(MotionEvent event1, MotionEvent event2,
      float velocityX, float velocityY)
{
      velocityX = velocityX > 2000 ? 2000 : velocityX;
      velocityX = velocityX < -2000 ? -2000 : velocityX;
      velocityY = velocityY > 2000 ? 2000 : velocityY;
      velocityY = velocityY < -2000 ? -2000 : velocityY;
      // 根据横向上的速度计算沿 Y 轴旋转的角度
      angley += velocityX * ROTATE_FACTOR / 4000;
      // 根据纵向上的速度计算沿 X 轴旋转的角度
      anglex += velocityY * ROTATE_FACTOR / 4000;
      return true;
}
@Override
public boolean onDown(MotionEvent arg0)
{
      return false;
}
@Override
public void onLongPress(MotionEvent event)
{
}
@Override
public boolean onScroll(MotionEvent event1, MotionEvent event2,
      float distanceX, float distanceY)
{
      return false;
}
@Override
public void onShowPress(MotionEvent event)
{
}
@Override
public boolean onSingleTapUp(MotionEvent event)
{
      return false;
}
public class MyRenderer implements Renderer
```

```
{
    // 立方体的顶点坐标（一共是 36 个顶点，组成 12 个三角形）
    private float[] cubeVertices = { -0.6f, -0.6f, -0.6f, 0.6f,
        -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, -0.6f, -0.6f,
        -0.6f, -0.6f, -0.6f, -0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f,
        0.6f, 0.6f, 0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, -0.6f, -0.6f,
        0.6f, -0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, 0.6f,
        0.6f, -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, -0.6f, 0.6f,
        -0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, 0.6f, 0.6f, 0.6f,
        0.6f, 0.6f, -0.6f, 0.6f, 0.6f, -0.6f, -0.6f, 0.6f, 0.6f, -0.6f,
        -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, 0.6f, -0.6f, 0.6f, 0.6f, 0.6f,
        0.6f, 0.6f, 0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, -0.6f,
        -0.6f, -0.6f, -0.6f, 0.6f, -0.6f, -0.6f, 0.6f, -0.6f, 0.6f, 0.6f,
        -0.6f, 0.6f, -0.6f, };
    // 定义立方体所需要的 6 个面（一共是 12 个三角形所需的顶点）
    private byte[] cubeFacets = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
        13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
        30, 31, 32, 33, 34, 35, };
    // 定义纹理贴图的坐标数据
    private float[] cubeTextures = { 1.0000f, 1.0000f, 1.0000f, 0.0000f,
        0.0000f, 0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f, 1.0000f,
        1.0000f, 0.0000f, 1.0000f, 1.0000f, 1.0000f, 1.0000f, 0.0000f,
        1.0000f, 0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f, 0.0000f,
        1.0000f, 1.0000f, 1.0000f, 1.0000f, 0.0000f, 1.0000f, 0.0000f,
        0.0000f, 0.0000f, 0.0000f, 1.0000f, 0.0000f, 1.0000f, 1.0000f,
        1.0000f, 1.0000f, 0.0000f, 1.0000f, 0.0000f, 0.0000f, 0.0000f,
        0.0000f, 1.0000f, 0.0000f, 1.0000f, 1.0000f, 1.0000f, 1.0000f,
        0.0000f, 1.0000f, 0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f,
        0.0000f, 1.0000f, 1.0000f, 1.0000f, 1.0000f, 0.0000f, 1.0000f,
        0.0000f, 0.0000f, 0.0000f, 0.0000f, 1.0000f };
    private Context context;
    private FloatBuffer cubeVerticesBuffer;
    private ByteBuffer cubeFacetsBuffer;
    private FloatBuffer cubeTexturesBuffer;
    // 定义本程序所使用的纹理
    private int texture;
    public MyRenderer(Context main)
    {
        this.context = main;
        // 将立方体的顶点位置数据数组包装成 FloatBuffer;
        cubeVerticesBuffer = floatBufferUtil(cubeVertices);
        // 将立方体的 6 个面（12 个三角形）的数组包装成 ByteBuffer
        cubeFacetsBuffer = ByteBuffer.wrap(cubeFacets);
        // 将立方体的纹理贴图的坐标数据包装成 FloatBuffer
        cubeTexturesBuffer = floatBufferUtil(cubeTextures);
    }
    @Override
    public void onSurfaceCreated(GL10 gl, EGLConfig config)
    {
        // 该方法的大部分代码与前面的示例相同，故省略
        ...
        // 启用 2D 纹理贴图
        gl.glEnable(GL10.GL_TEXTURE_2D);
        // 装载纹理
        loadTexture(gl);
    }
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height)
    {
        // 该方法的大部分代码与前面的示例相同，故省略
        ...
    }
```

```
public void onDrawFrame(GL10 gl)
{
    // 清除屏幕缓存和深度缓存
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
    // 启用顶点坐标数据
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
    // 启用贴图坐标数组数据
    gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);  //①
    // 设置当前矩阵模式为模型视图。
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    // 把绘图中心移入屏幕 2 个单位
    gl.glTranslatef(0f, 0.0f, -2.0f);
    // 旋转图形
    gl.glRotatef(angley, 0, 1, 0);
    gl.glRotatef(anglex, 1, 0, 0);
    // 设置顶点的位置数据
    gl.glVertexPointer(3, GL10.GL_FLOAT, 0, cubeVerticesBuffer);
    // 设置贴图的坐标数据
    gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, cubeTexturesBuffer);//②
    // 执行纹理贴图
    gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);  //③
    // 按 cubeFacetsBuffer 指定的面绘制三角形
    gl.glDrawElements(GL10.GL_TRIANGLES, cubeFacetsBuffer.
    remaining(),
        GL10.GL_UNSIGNED_BYTE, cubeFacetsBuffer);
    // 绘制结束
    gl.glFinish();
    // 禁用顶点、纹理坐标数组
    gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
    gl.glDisableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
    // 递增角度值以便每次以不同角度绘制
}
private void loadTexture(GL10 gl)
{
    Bitmap bitmap = null;
    try
    {
        // 加载位图
        bitmap = BitmapFactory.decodeResource(context.getResources(),
            R.drawable.sand);
        int[] textures = new int[1];
        // 指定生成 N 个纹理（第一个参数指定生成一个纹理）
        // textures 数组将负责存储所有纹理的代号
        gl.glGenTextures(1, textures, 0);
        // 获取 textures 纹理数组中的第一个纹理
        texture = textures[0];
        // 通知 OpenGL 将 texture 纹理绑定到 GL10.GL_TEXTURE_2D 目标中
        gl.glBindTexture(GL10.GL_TEXTURE_2D, texture);
        // 设置纹理被缩小（距离视点很远时被缩小）时的滤波方式
        gl.glTexParameterf(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_NEAREST);
        // 设置纹理被放大（距离视点很近时被方法）时的滤波方式
        gl.glTexParameterf(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
        // 设置在横向、纵向上都是平铺纹理
        gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_
        WRAP_S,
            GL10.GL_REPEAT);
        gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_
        WRAP_T,
            GL10.GL_REPEAT);
```

```
                    // 加载位图生成纹理
                    GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
                }
                finally
                {
                    // 生成纹理之后，回收位图
                    if (bitmap != null)
                        bitmap.recycle();
                }
            }
        }
        // 省略 floatBufferUtil 工具方法
        ...
    }
```

上面的程序中①、②、③号代码就完成纹理贴图的三个步骤。

程序中粗体字代码用于计算用户手势在 *X* 方向、*Y* 方向上速度，并根据 *X* 方向、*Y* 方向上的速度来改变立方体的旋转角度，这样就可以让该立方体随用户的手势而转动了。

可能有读者会发现上面这个立方体的顶点坐标看上去很“可怕”，怎么一个立方体需要这么多顶点呢？实际上是因为笔者实在不想在纸上先画这个立方体、再数立方体的每个三角面由哪些顶点组成，再计算每个三角面的贴图坐标——这太耗时间了。笔者直接使用 3D MAX 建了一个立方体模型，并从中导出了模型的每个顶点的位置信息、每个三角面由哪些顶点组成，以及贴图的坐标信息。

3D MAX 就不会像我们之前那样“复用”立方体的顶点——它直接计算该立方体需要 12 个三角面，每个三角面需要 3 个顶点，这样一共是 36 个顶点——其实有大量顶点的位置是相同，但 3D MAX 不管这些。它认为这个立方体需要 36 个顶点，每个顶点的位置需要 *X*、*Y*、*Z* 三个坐标值，因此导出这个立方体的顶点坐标信息的数组就是一个长度为 108 的数组。



提示：

由于本书只是介绍简单的 OpenGL ES 编程，因此涉及 3D MAX 的内容就此打住，如果读者还有更多兴趣，则可以参考 3D MAX 的相关资料。

运行上面的程序，用户将可以看到一个具有贴图纹理的三维立方体，用户可以通过拖动手势来旋转该立方体，程序效果如图 12.11 所示。

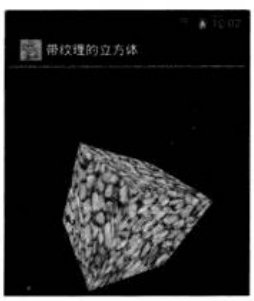


图 12.11　具有贴图的立方体

需要指出的是，OpenGL ES 虽然只是 OpenGL 的一个子集，但其实它所包含的功能也很多，而本书的重点是介绍 Android 所支持的 3D 开发，限于篇幅，不可能深入介绍 OpenGL ES 编程的全部内容，如果读者希望深入了解 OpenGL 编程的内容，可以自行参考相关书籍和资料。

12.5　本章小结

本章主要介绍了 Android 3D 编程的入门知识，简要介绍了 2D 绘图与 3D 绘图的联系与区别，也简要介绍了 OpenGL 与 OpenGL ES，Android 系统内置了对 OpenGL ES 的支持，开发者可以在 Android 应用中通过 OpenGL ES 来绘制 3D 图形。学习本章需要掌握 3D 绘图的基本理论、三维坐标系统等。除此之外，还需要重点掌握如何在 Android 环境中使用 Open GL ES，以及通过 OpenGL ES 绘制 2D 图形、3D 图形的方法。

CHAPTER

13

第13章 Android网络应用

本章要点

- TCP 协议的基础
- 使用 ServerSocket 建立服务器
- 使用 Socket 进行网络通信
- 在网络通信中加入多线程支持
- 使用 URL 读取网络资源
- 使用 URLConnection 提交请求
- 使用 HttpURLConnection
- Apache HttpClient 的基础知识
- 使用 HttpClient 维护用户状态、发送请求、获取响应
- 使用 WebView 浏览网页
- 使用 WebView 加载、显示 HTML 代码
- 使用 WebView 中 JavaScript 脚本调用 Android 方法
- Web Service 的基本知识
- Web Service 平台简介
- 使用 ksoap2-android 项目来调用远程 Web Service

手机本身是作为手持终端来使用的，因此它的计算能力、存储能力都是有限的。它的主要优势是携带方便，可以随时打开，而且手机通常总是处于联网状态。因此网络支持对于手机应用的重要性不言而喻。

Android 完全支持 JDK 本身的 TCP、UDP 网络通信 API，也可以使用 ServerSocket、Socket 来建立基于 TCP/IP 协议的网络通信；也可以使用 DatagramSocket、Datagrampacket、MulticastSocket 来建立基于 UDP 协议的网络通信。如果读者有 Java 网络编程的经验，这些经验完全适用于 Android 应用的网络编程。Android 也支持 JDK 提供的 URL、URLConnection 等网络通信 API。



提示：限于篇幅，本章并未涉及 UDP 协议编程的相关内容，如果读者需要掌握基于 UDP 协议，使用 DatagramSocket、Datagrampacket、MulticastSocket 进行网络编程的内容，可以参考疯狂 Java 体系的《疯狂 Java 讲义》。

不仅如此，Android 还内置了 HttpClient，这样可以非常方便地发送 HTTP 请求，并获取 HTTP 响应，通过内置 HttpClient，Android 简化了与网站之间的交互。令人遗憾的是，Android 并未内置对 Web Service 的支持，为了弥补这种不足，本章将会介绍如何利用 ksoap2-android 项目在 Android 应用中调用远程 Web Service。

13.1　基于 TCP 协议的网络通信

TCP/IP 通信协议是一种可靠的网络协议，它在通信的两端各建立一个 Socket，从而在通信的两端之间形成网络虚拟链路。一旦建立了虚拟的网络链路，两端的程序就可以通过虚拟链路进行通信。Java 对基于 TCP 协议的网络通信提供了良好的封装，Java 使用 Socket 对象来代表两端的通信接口，并通过 Socket 产生 I/O 流来进行网络通信。

➤➤13.1.1　TCP 协议基础

IP 协议是 Internet 上使用的一个关键协议，它的全称是 Internet Protocol，即 Internet 协议，通常简称 IP 协议。通过使用 IP 协议，使 Internet 成为一个允许连接不同类型的计算机和不同操作系统的网络。

要使两台计算机彼此之间进行通信，必须使两台计算机使用同一种“语言”，IP 协议只保证计算机能发送和接收分组数据。IP 协议负责将消息从一个主机传送到另一个主机，消息在传送的过程中被分割成一个个小包。

尽管计算机通过安装 IP 软件，保证了计算机之间可以发送和接收数据，但 IP 协议还不能解决数据分组在传输过程中可能出现的问题。因此，若要解决可能出现的问题，连接上 Internet 的计算机还需要安装 TCP 协议来提供可靠并且无差错的通信服务。

TCP 协议被称为一种端对端协议。这是因为它为两台计算机之间的连接起了重要作用：当一台计算机需要与另一台远程计算机连接时，TCP 协议会让它们建立一个连接：用于发送和接收数据的虚拟链路。

TCP 协议负责收集这些信息包，并将其按适当的次序放好传送，在接收端收到后再将其正确地还原。TCP 协议保证了数据包在传送中准确无误。TCP 协议使用重发机制：当一个通

信实体发送一个消息给另一个通信实体后，需要收到另一个通信实体的确认信息，如果没有收到另一个通信实体的确认信息，则会再次重发刚才发送的信息。

通过这种重发机制，TCP 协议向应用程序提供可靠的通信连接，使它能够自动适应网上的各种变化。即使在 Internet 暂时出现堵塞的情况下，TCP 也能够保证通信的可靠。

图 13.1 显示了 TCP 协议控制两个通信实体互相通信的示意图。

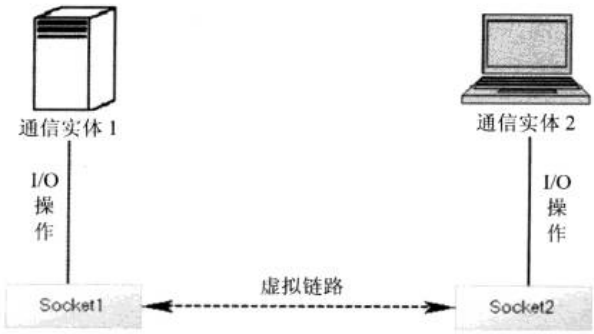


图 13.1 TCP 协议的通信示意图

综上所述，虽然 IP 和 TCP 这两个协议的功能不尽相同，也可以分开单独使用，但它们是在同一时期作为一个协议来设计的，并且在功能上也是互补的。只有两者结合，才能保证 Internet 在复杂的环境下正常运行。凡是要连接到 Internet 的计算机，都必须同时安装和使用这两个协议，因此在实际中常把这两个协议统称为 TCP/IP 协议。

13.1.2 使用 ServerSocket 创建 TCP 服务器端

从图 13.1 中看上去 TCP 通信的两个通信实体之间并没有服务器端、客户端之分，但那是两个通信实体已经建立虚拟链路之后的示意图。在两个通信实体没有建立虚拟链路之前，必须有一个通信实体先做出“主动姿态”，主动接收来自其他通信实体的连接请求。

Java 中能接收其他通信实体连接请求的类是 ServerSocket，ServerSocket 对象用于监听来自客户端的 Socket 连接，如果没有连接，它将一直处于等待状态。ServerSocket 包含一个监听来自客户端连接请求的方法。

- Socket accept()：如果接收到一个客户端 Socket 的连接请求，该方法将返回一个与连接客户端 Socket 对应的 Socket（如图 13.1 所示，每个 TCP 连接有两个 Socket）；否则该方法将一直处于等待状态，线程也被阻塞。

为了创建 ServerSocket 对象，ServerSocket 类提供了如下几个构造器。

- ServerSocket(int port)：用指定的端口 port 来创建一个 ServerSocket。该端口应该是有一个有效的端口整数值：0～65 535。
- ServerSocket(int port,int backlog)：增加一个用来改变连接队列长度的参数 backlog。
- ServerSocket(int port,int backlog,InetAddress localAddr)：在机器存在多个 IP 地址的情况下，允许通过 localAddr 这个参数来指定将 ServerSocket 绑定到指定的 IP 地址。

当 ServerSocket 使用完毕后，应使用 ServerSocket 的 close()方法来关闭该 ServerSocket。通常情况下，服务器不应该只接收一个客户端请求，而应该不断地接收来自客户端的所有请

求，所以 Java 程序通常会通过循环不断地调用 ServerSocket 的 accept()方法，如以下代码片段所示。

```
// 创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
ServerSocket ss = new ServerSocket(30000);
// 采用循环不断接收来自客户端的请求
while (true)
{
    // 每当接收到客户端 Socket 的请求，服务器端也对应产生一个 Socket
    Socket s = ss.accept();
    // 下面就可以使用 Socket 进行通信了
    ...
}
```



提示：

上面的程序中创建 ServerSocket 没有指定 IP 地址，则该 ServerSocket 将会绑定到本机默认的 IP 地址。程序中使用 30000 作为该 ServerSocket 的端口号，通常推荐使用 1024 以上的端口，主要是为了避免与其他应用程序的通用端口冲突。

由于手机无线上网的 IP 地址通常都是由移动运营公司动态分配的，一般不会有自己固定的 IP 地址，因此很少在手机上运行服务器端，服务器端通常运行在有固定 IP 的服务器上。本节所介绍的应用程序的服务器端也是运行在 PC 上的。

13.1.3 使用 Socket 进行通信

客户端通常可使用 Socket 的构造器来连接到指定服务器，Socket 通常可使用如下两个构造器。

- Socket(InetAddress/String remoteAddress, int port)：创建连接到指定远程主机、远程端口的 Socket，该构造器没有指定本地地址、本地端口，默认使用本地主机的默认 IP 地址，默认使用系统动态指定的 IP 地址。
- Socket(InetAddress/String remoteAddress, int port, InetAddress localAddr, int localPort)：：创建连接到指定远程主机、远程端口的 Socket，并指定本地 IP 地址和本地端口号，适用于本地主机有多个 IP 地址的情形。

上面两个构造器中指定远程主机时既可使用 InetAddress 来指定，也可直接使用 String 对象来指定，但程序通常使用 String 对象（如 192.168.2.23）来指定远程 IP。当本地主机只有一个 IP 地址时，使用第一个方法更为简单。如以下代码所示：

```
// 创建连接到本机、30000 端口的 Socket
Socket s = new Socket("192.168.1.88" , 30000);
// 下面就可以使用 Socket 进行通信了
...
```

当程序执行上面代码中的粗体字代码时，该代码将会连接到指定服务器，让服务器端的 ServerSocket 的 accept()方法向下执行，于是服务器端和客户端就产生一对互相连接的 Socket。



提示：

上面的程序连接到“远程主机”的 IP 地址使用的是 192.168.1.88，这个 IP 地址就是笔者运行服务器端程序的 IP 地址。

当客户端、服务器端产生了对应的 Socket 之后，此时就到了如图 13.1 所示的通信示意图，程序无须再区分服务器、客户端，而是通过各自的 Socket 进行通信。Socket 提供如下两个方法来获取输入流和输出流。

- InputStream getInputStream()：返回该 Socket 对象对应的输入流，让程序通过该输入流从 Socket 中取出数据。
- OutputStream getOutputStream()：返回该 Socket 对象对应的输出流，让程序通过该输出流向 Socket 中输出数据。

看到这两个方法返回的 InputStream 和 OutputStream，读者应该可以明白 Java 在设计 I/O 体系上的苦心了：不管底层的 I/O 流是怎样的节点流：文件流也好，网络 Socket 产生的流也好，程序都可以将其包装成处理流，从而提供更多方便的处理。下面以一个最简单的网络通信程序为例来介绍基于 TCP 协议的网络通信。



提示：

如果读者朋友阅读过疯狂 Java 体系的《疯狂 Java 讲义》，可能会发现本章的示例程序、文字内容与《疯狂 Java 讲义》介绍网络编程的一章有较多相同的内容。实际上也是这样的，因为 Android 网络通信还是依赖于 JDK 在 java.net、java.io 两个包下的 API。

下面的服务器程序需要在 PC 上运行，该程序非常简单，因此不需要建立 Android 项目，直接定义一个 Java 类，并运行该 Java 类即可。它仅仅建立 ServerSocket 监听，并使用 Socket 获取输出流输出。

程序清单：codes\13\13.1\SimpleServer\SimpleServer.java

```
public class SimpleServer
{
    public static void main(String[] args)
        throws IOException
    {
        // 创建一个 ServerSocket，用于监听客户端 Socket 的连接请求
        ServerSocket ss = new ServerSocket(30000);  //①
        // 采用循环不断接收来自客户端的请求
        while (true)
        {
            // 每当接收到客户端 Socket 的请求，服务器端也对应产生一个 Socket
            Socket s = ss.accept();
            OutputStream os = s.getOutputStream();
            os.write("您好，您收到了服务器的新年祝福！\n"
                .getBytes("utf-8"));
            // 关闭输出流，关闭 Socket
            os.close();
            s.close();
        }
    }
}
```

上面的程序中①号代码建立了一个 ServerSocket，该 ServerSocket 在 30000 端口监听，该 ServerSocket 将会一直监听，等待客户端程序的连接。程序接下来的 4 行粗体字代码用于打开 Socket 对应输出流，并向输出流中写入一段字符串数据。

注意：

上面的程序并未把 OutputStream 流包装成 PrintStream，然后使用 PrintStream 直接输出整个字符串，这是因为该服务器端程序运行于 Windows 主机上，当直接使用 PrintStream 输出字符串时默认使用系统平台的字符串（即 GBK）进行编码；但该程序的客户端是 Android 应用，运行于 Linux 平台（Android 是 Linux 内核的），因此当客户端读取网络数据时默认使用 UTF-8 字符集进行解码，这样势必引起乱码。为了保证客户端能正常解析到数据，此处手动控制字符串的编码，强行指定使用 UTF-8 字符集进行编码，这样就可以避免乱码问题了。



接下来的客户端程序也非常简单，它仅仅使用 Socket 建立与指定 IP、指定端口的连接，并使用 Socket 获取输入流读取数据。该客户端程序是一个 Android 应用，因此还是需要先建立 Android 项目，该程序的界面中包含一个文本框，用于显示从服务器端读取的字符串数据。

程序清单：codes\13\13.1\SimpleClient\src\org\crazyit\net\SimpleClient.java

```
public class SimpleClient extends Activity
{
    EditText show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (EditText) findViewById(R.id.show);
        new Thread()
        {
            @Override
            public void run()
            {
                try
                {
                    // 建立连接到远程服务器的 Socket
                    Socket socket = new Socket("192.168.1.88" , 30000);//①
                    // 将 Socket 对应的输入流包装成 BufferedReader
                    BufferedReader br = new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
                    // 进行普通 I/O 操作
                    String line = br.readLine();
                    show.setText("来自服务器的数据：" + line);
                    // 关闭输入流、socket
                    br.close();
                    socket.close();
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
        }.start();
    }
}
```

上面的程序中①号粗体字代码是使用 ServerSocket 和 Socket 建立网络连接的代码，接下来的几行粗体字代码是通过 Socket 获取输入流、输出流进行通信的代码。通过程序不难看出，一旦使用 ServerSocket、Socket 建立网络连接之后，程序通过网络通信与普通 IO 并没有太大的区别。

该程序与本书第 1 版的程序有所区别：本书第 1 版为了简化编程、突出网络编程的主题，因此直接在 UI 线程中建立网络连接，通过网络读取服务器响应数据——但实际项目中并不推荐这么做，因为建立网络连接、网络通信是不稳定的，它所需要的时间也不确定，因此直接在 UI 线程中建立网络连接、通过网络读取数据可能阻塞 UI 线程，导致 Android 应用失去响应。在最新的 Android 平台上，Android 已经不允许在 UI 线程中建立网络连接，因此上面的示例启动了一条新线程来建立网络连接，并读取网络数据。

注意

新版 Android 平台与 Android 2.3.x 在网络编程这一章最大的区别在于：新版 Android 平台不允许直接在 UI 线程建立网络连接、访问网络资源，因此本章后面的所有示例程序都会将建立网络连接、访问网络资源的操作放在新线程中完成。



该 Android 应用需要访问互联网，因此还需要为该应用赋予访问互联网的权限，也就是在 AndroidManifest.xml 文件中增加如下配置片段：

```
<!-- 授权访问互联网-->
<uses-permission android:name="android.permission.INTERNET"/>
```

先运行上面程序中的 SimpleServer 类，将看到服务器一直处于等待状态，因为服务器使用了死循环来接收来自客户端的请求；再运行客户端 AndroidClient 类，将可看到程序输出：“来自服务器的数据：您好，您收到了服务器的新年祝福！”，如图 13.2 所示，这表明客户端和服务器端通信成功。



图 13.2 与远程服务器交互

上面的程序为了突出通过 ServerSocket 和 Socket 建立连接并通过底层 IO 流进行通信的主题，程序没有进行异常处理，也没有使用 finally 块来关闭资源。

实际应用中，程序可能不想让执行网络连接、读取服务器数据的进程一直阻塞，而是希望当网络连接、读取操作超过合理时间之后，系统自动认为该操作失败，这个合理时间就是超时时长。Socket 对象提供了一个 setSoTimeout(int timeout)来设置超时时长，如下面的代码片段所示：

```
Socket s = new Socket("127.0.0.1" , 30000);
// 设置 10 秒之后即认为超时
s.setSoTimeout(10000);
```

为 Socket 对象指定了超时时长之后，如果使用 Socket 进行读、写操作完成之前已经超出了该时间限制，那么这些方法就会抛出 SocketTimeoutException 异常，程序可以对该异常进行捕捉，并进行适当处理，如以下代码所示。

```
try
{
    // 使用 Scanner 来读取网络输入流中的数据
    Scanner scan = new Scanner(s.getInputStream())
```

```
        // 读取一行字符
        String line = scan.nextLine()
        ...
    }
    // 捕捉 SocketTimeoutException 异常
    catch(SocketTimeoutException ex)
    {
        // 对异常进行处理
        ...
    }
```

假设程序需要为 Socket 连接服务器时指定超时时长：即经过指定时间后，如果该 Socket 还未连接到远程服务器，则系统认为该 Socket 连接超时。但 Socket 的所有构造器里都没有提供指定超时时长的参数，所以程序应该先创建一个无连接的 Socket，再调用 Socket 的 connect() 方法来连接远程服务器，而 connect()方法就可以接受一个超时时长参数。如以下代码所示：

```
// 创建一个无连接的 Socket
Socket s = new Socket();
// 让该 Socket 连接到远程服务器，如果经过 10 秒还没有连接到，则认为连接超时
s.connect(new InetAddress(host, port) ,10000);
```

13.1.4 加入多线程

前面服务器端和客户端只是进行了简单的通信操作：服务器接收到客户端连接之后，服务器向客户端输出一个字符串，而客户端也只是读取服务器的字符串后就退出了。实际应用中的客户端则可能需要和服务器端保持长时间通信，即服务器需要不断地读取客户端数据，并向客户端写入数据；客户端也需要不断地读取服务器数据，并向服务器写入数据。

当使用传统 BufferedReader 的 readLine()方法读取数据时，当该方法成功返回之前，线程被阻塞，程序无法继续执行。考虑到这个原因，服务器应该为每个 Socket 单独启动一条线程，每条线程负责与一个客户端进行通信。

客户端读取服务器数据的线程同样会被阻塞，所以系统应该单独启动一条线程，该线程专门负责读取服务器数据。

下面考虑实现一个简单的 C/S 聊天室应用，服务器端则应该包含多条线程，每个 Socket 对应一条线程，该线程负责读取 Socket 对应输入流的数据（从客户端发送过来的数据），并将读到的数据向每个 Socket 输出流发送一遍（将一个客户端发送的数据“广播”给其他客户端），因此需要在服务器端使用 List 来保存所有的 Socket。

下面是服务器端的实现代码，程序为服务器提供了两个类，一个是创建 ServerSocket 监听的主类，另一个是负责处理每个 Socket 通信的线程类。

程序清单：codes\13\13.1\MultiThreadServer\MyServer.java

```
public class MyServer
{
    // 定义保存所有 Socket 的 ArrayList
    public static ArrayList<Socket> socketList
        = new ArrayList<Socket>();
    public static void main(String[] args)
        throws IOException
    {
        ServerSocket ss = new ServerSocket(30000);
        while(true)
        {
```

```
                // 此行代码会阻塞，将一直等待别人的连接
                Socket s = ss.accept();
                socketList.add(s);
                // 每当客户端连接后启动一条 ServerThread 线程为该客户端服务
                new Thread(new ServerThread(s)).start();
            }
        }
    }
```

上面的程序是服务器端只负责接收客户端 Socket 的连接请求，每当客户端 Socket 连接到该 ServerSocket 之后，程序将对应 Socket 加入 socketList 集合中保存，并为该 Socket 启动一条线程，该线程负责处理该 Socket 所有的通信任务，如程序中 4 行粗体字代码所示。服务器端线程类的代码如下。

程序清单：codes\13\13.1\MultiThreadServer\ServerThread.java

```
// 负责处理每个线程通信的线程类
public class ServerThread implements Runnable
{
    // 定义当前线程所处理的 Socket
    Socket s = null;
    // 该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    public ServerThread(Socket s)
        throws IOException
    {
        this.s = s;
        // 初始化该 Socket 对应的输入流
        br = new BufferedReader(new InputStreamReader(
            s.getInputStream() , "utf-8"));   //②
    }
    public void run()
    {
        try
        {
            String content = null;
            // 采用循环不断从 Socket 中读取客户端发送过来的数据
            while ((content = readFromClient()) != null)
            {
                // 遍历 socketList 中的每个 Socket，
                // 将读到的内容向每个 Socket 发送一次
                for (Socket s : MyServer.socketList)
                {
                    OutputStream os = s.getOutputStream();
                    os.write((content + "\n").getBytes("utf-8"));
                }
            }
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    // 定义读取客户端数据的方法
    private String readFromClient()
    {
        try
        {
            return br.readLine();
        }
        // 如果捕捉到异常，表明该 Socket 对应的客户端已经关闭
```

```
            catch (IOException e)
            {
                // 删除该 Socket
                MyServer.socketList.remove(s);    //①
            }
            return null;
        }
    }
```

上面的服务器端线程类不断读取客户端数据，程序使用 readFromClient()方法来读取客户端数据，如果读取数据过程中捕获到 IOException 异常，则表明该 Socket 对应的客户端 Socket 出现了问题（到底什么问题我们不管，反正不正常），程序就将该 Socket 从 socketList 中删除，如 readFromClient()方法中①号代码所示。

当服务器线程读到客户端数据之后，程序遍历 socketList 集合，并将该数据向 socketList 集合中的每个 Socket 发送一次——该服务器线程将把从 Socket 中读到的数据向 socketList 中的每个 Socket 转发一次，如 run()线程执行体中的粗体字代码所示。



提示：

上面的程序中②号粗体字代码将网络的字节输入流转换为字符输入流时，指定了转换所用的字符串：UTF-8，这也是由于客户端写过来的数据是采用 UTF-8 字符集进行编码的，所以此处的服务器端也要使用 UTF-8 字符集进行解码。当需要编写跨平台的网络通信程序时，使用 UTF-8 字符集进行编码、解码是一种较好的解决方案。

每个客户端应该包含两条线程：一条负责生成主界面，并响应用户动作，并将用户输入的数据写入 Socket 对应的输出流中；另一条负责读取 Socket 对应输入流中的数据（从服务器发送过来的数据），并负责将这些数据在程序界面上显示出来。

客户端程序同样是一个 Android 应用，因此需要创建一个 Android 项目，这个 Android 应用的界面中包含两个文本框：一个用于接收用户输入，另一个用于显示聊天信息；界面中还有一个按钮，当用户单击该按钮时，程序向服务器发送聊天信息。该程序的界面布局代码如下。

程序清单：codes\13\13.1\MultiThreadClient\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    >
<!-- 定义一个文本框，它用于接收用户的输入 -->
<EditText
    android:id="@+id/input"
    android:layout_width="240dp"
    android:layout_height="wrap_content"
    />
<Button
    android:id="@+id/send"
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="8px"
        android:text="@string/send"
        />
</LinearLayout>
<!-- 定义一个文本框，它用于显示来自服务器的信息 -->
<TextView
        android:id="@+id/show"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="top"
        android:background="#ffff"
        android:textSize="14dp"
        android:textColor="#f000"
        />
</LinearLayout>
```

客户端的 Activity 负责生成程序界面，并为程序的按钮单击事件绑定事件监听器，当用户单击按钮时向服务器发送信息。客户端的 Activity 代码如下。

程序清单：codes\13\13.1\MultiThreadClient\src\org\crazyit\net\MultiThreadClient.java

```
public class MultiThreadClient extends Activity
{
    // 定义界面上的两个文本框
    EditText input;
    TextView show;
    // 定义界面上的一个按钮
    Button send;
    Handler handler;
    // 定义与服务器通信的子线程
    ClientThread clientThread;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        input = (EditText) findViewById(R.id.input);
        send = (Button) findViewById(R.id.send);
        show = (TextView) findViewById(R.id.show);
        handler = new Handler() //②
        {
            @Override
            public void handleMessage(Message msg)
            {
                // 如果消息来自于子线程
                if (msg.what == 0x123)
                {
                    // 将读取的内容追加显示在文本框中
                    show.append("\n" + msg.obj.toString());
                }
            }
        };
        clientThread = new ClientThread(handler);
        // 客户端启动 ClientThread 线程创建网络连接、读取来自服务器的数据
        new Thread(clientThread).start(); //①
        send.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
```

```
                        try
                        {
                            // 当用户按下发送按钮后，将用户输入的数据封装成 Message
                            // 然后发送给子线程的 Handler
                            Message msg = new Message();
                            msg.what = 0x345;
                            msg.obj = input.getText().toString();
                            clientThread.revHandler.sendMessage(msg);
                            // 清空 input 文本框
                            input.setText("");
                        }
                        catch (Exception e)
                        {
                            e.printStackTrace();
                        }
                    }
                });
        }
}
```

当用户单击该程序界面中的“发送”按钮之后，程序将会把 input 输入框中的的内容发送该 clientThread 的 revHandler 对象，clientThread 将负责将用户输入的内容发送给服务器。

为了避免 UI 线程被阻塞，该程序将建立网络连接、与网络服务器通信等工作都交给 ClientThread 线程完成。因此该程序在①号代码处启动 ClientThread 线程。

由于 Android 不允许子线程访问界面组件，因此上面的程序定义了一个 Handler 来处理来自子线程的消息，如程序中②号粗体字代码所示。

ClientThread 子线程负责建立与远程服务器的连接，并负责与远程服务器通信，读到数据之后便通过 Handler 对象发送一条消息；当 ClientThread 子线程收到 UI 线程发送过来的消息（消息携带了用户输入的内容）之后，还负责将用户输入的内容发送给远程服务器。该子线程代码如下。

程序清单：codes\13\13.1\MultiThreadClient\src\org\crazyit\net\ClientThread.java

```
public class ClientThread implements Runnable
{
    private Socket s;
    // 定义向 UI 线程发送消息的 Handler 对象
    private Handler handler;
    // 定义接收 UI 线程的消息的 Handler 对象
    public Handler revHandler;
    // 该线程所处理的 Socket 所对应的输入流
    BufferedReader br = null;
    OutputStream os = null;
    public ClientThread(Handler handler)
    {
        this.handler = handler;
    }
    public void run()
    {
        try
        {
            s = new Socket("192.168.1.88", 30000);
            br = new BufferedReader(new InputStreamReader(
                s.getInputStream()));
            os = s.getOutputStream();
            // 启动一条子线程来读取服务器响应的数据
            new Thread()
            {
```

```
                @Override
                public void run()
                {
                    String content = null;
                    // 不断读取 Socket 输入流中的内容
                    try
                    {
                        while ((content = br.readLine()) != null)
                        {
                            // 每当读到来自服务器的数据之后，发送消息通知程序
                            // 界面显示该数据
                            Message msg = new Message();
                            msg.what = 0x123;
                            msg.obj = content;
                            handler.sendMessage(msg);
                        }
                    }
                    catch (IOException e)
                    {
                        e.printStackTrace();
                    }
                }
            }.start();
            // 为当前线程初始化 Looper
            Looper.prepare();
            // 创建 revHandler 对象
            revHandler = new Handler()
            {
                @Override
                public void handleMessage(Message msg)
                {
                    // 接收到 UI 线程中用户输入的数据
                    if (msg.what == 0x345)
                    {
                        // 将用户在文本框内输入的内容写入网络
                        try
                        {
                            os.write((msg.obj.toString() + "\r\n")
                                .getBytes("utf-8"));
                        }
                        catch (Exception e)
                        {
                            e.printStackTrace();
                        }
                    }
                }
            };
            // 启动 Looper
            Looper.loop();
        }
        catch (SocketTimeoutException e1)
        {
            System.out.println("网络连接超时!! ");
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

上面线程的功能也非常简单，它只是不断获取 Socket 输入流中的内容，当读到 Socket 输入流中的内容后，便通过 Handler 对象发送一条消息，消息负责携带读到的数据，如上面程序中第一段粗体字代码所示；除此之外，该子线程还负责读取 UI 线程发送的消息，接收到消息之后，该子线程负责将消息中携带的数据发送给远程服务器，如上面程序中第二段粗体字代码所示。

先运行上面程序中的 MyServer 类，该类运行后只是作为服务器，看不到任何输出。接着可以运行 Android 客户端——相当于启动聊天室客户端登录该服务器，接着可以看到在任何一个 Android 客户端输入一些内容后单击“发送”按钮，将可看到所有客户端（包括自己）都会收到他刚刚输入的内容，如图 13.3 所示，这就粗略实现了一个 C/S 结构聊天室的功能。



图 13.3　支持多线程的 TCP 客户端

借助于此处介绍的网络通信机制，我们可以在 Android 平台上开发大量功能强大的网络通信程序，比如《疯狂 Java 讲义》中所介绍的网络五子棋、网络斗地主等，这些程序的网络通信部分都可按上面介绍的方式来实现，只是不同游戏可能需要在网络上交换不同类型的数据，这可能需要封装自己的网络协议，关于基于 Socket 通信的更详细介绍，读者可以对照《疯狂 Java 讲义》中网络编程的相关知识，这些知识是完全相通的。

13.2　使用 URL 访问网络资源

URL（Uniform Resource Locator）对象代表统一资源定位器，它是指向互联网“资源”的指针。资源可以是简单的文件或目录，也可以是对更复杂的对象的引用，例如对数据库或搜索引擎的查询。通常情况而言，URL 可以由协议名、主机、端口和资源组成。即满足如下格式：

```
protocol://host:port/resourceName
```

例如如下的 URL 地址：

```
http://www.crazyit.org/index.php
```



提示：

JDK 中还提供了一个 URI（Uniform Resource Identifiers）类，其实例代表一个统一资源标识符，Java 的 URIf 不能用于定位任何资源，它的唯一作用就是解析。与此对应的是，URL 则包含一个可打开到达该资源的输入流，因此我们可以将 URL 理解成 URI 的特例。

URL 类提供了多个构造器用于创建 URL 对象，一旦获得了 URL 对象之后，可以调用如下常用方法来访问该 URL 对应的资源。

- String getFile()：获取此 URL 的资源名。
- String getHost()：获取此 URL 的主机名。
- String getPath()：获取此 URL 的路径部分。
- int getPort()：获取此 URL 的端口号。
- String getProtocol()：获取此 URL 的协议名称。

- String getQuery()：获取此 URL 的查询字符串部分。
- URLConnection openConnection()：返回一个 URLConnection 对象，它表示到 URL 所引用的远程对象的连接。
- InputStream openStream()：打开与此 URL 的连接，并返回一个用于读取该 URL 资源的 InputStream。

13.2.1 使用 URL 读取网络资源

URL 对象中前面几个方法都非常容易理解，而该对象提供的 openStream()可以读取该 URL 资源的 InputStream，通过该方法可以非常方便地读取远程资源。

下面的程序示范如何通过 URL 类读取远程资源。

程序清单：codes\13\13.2\URLTest\src\org\crazyit\net\URLTest.java

```
public class URLTest extends Activity
{
    ImageView show;
    // 代表从网络下载得到的图片
    Bitmap bitmap;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if(msg.what == 0x123)
            {
                // 使用 ImageView 显示该图片
                show.setImageBitmap(bitmap);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show = (ImageView) findViewById(R.id.show);
        new Thread()
        {
            public void run()
            {
                try
                {
                    // 定义一个 URL 对象
                    URL url = new URL("http://www.crazyit.org/"
                        + "attachments/month_1008/20100812_7763e970f"
                        + "822325bfb019ELQVym8tW3A.png");
                    // 打开该 URL 对应的资源的输入流
                    InputStream is = url.openStream();
                    // 从 InputStream 中解析出图片
                    bitmap = BitmapFactory.decodeStream(is);
                    // 发送消息、通知 UI 组件显示该图片
                    handler.sendEmptyMessage(0x123);
                    is.close();
                    // 再次打开 URL 对应的资源的输入流
                    is = url.openStream();
                    // 打开手机文件对应的输出流
```

```
                        OutputStream os = openFileOutput("crazyit.png"
                                , MODE_WORLD_READABLE);
                        byte[] buff = new byte[1024];
                        int hasRead = 0;
                        // 将 URL 对应的资源下载到本地
                        while((hasRead = is.read(buff)) > 0)
                        {
                            os.write(buff, 0 , hasRead);
                        }
                        is.close();
                        os.close();
                    }
                    catch (Exception e)
                    {
                        e.printStackTrace();
                    }
                }
            }.start();
        }
    }
```

上面的程序两次调用了 URL 对象的 openStream()方法打开 URL 对应的资源的输入流，程序第一次使用 BitmapFactory 的 decodeStream(InputStream)方法来解析该输入流中的图片；第二次则使用 IO 将输入流中的图片下载到本地。

该程序同样需要访问互联网，因此需要授予该程序访问网络的权限，也就是需要在 AndroidManifest.xml 文件中增加如下授权代码：

```
<!-- 授权访问网络 -->
<uses-permission android:name="android.permission.INTERNET"/>
```

运行该程序将可以看到如图 13.4 所示的输出。

如图 13.4 所显示的图片就是程序中 URL 对象所对应的图片，运行该程序不仅可以显示该 URL 对象所对应的图片，而且还会在手机文件系统的/data/data/org.crazyit.net/files/目录下生成 crazyit.png 图片，该图片就是通过 URL 从网络上下载的图片。



图 13.4　使用 URL 读取网络图片

13.2.2　使用 URLConnection 提交请求

URL 的 openConnection()方法将返回一个 URLConnection 对象，该对象表示应用程序和 URL 之间的通信连接。程序可以通过 URLConnection 实例向该 URL 发送请求，读取 URL 引用的资源。

通常创建一个和 URL 的连接，并发送请求、读取此 URL 引用的资源需要如下几个步骤。

① 通过调用 URL 对象 openConnection()方法来创建 URLConnection 对象。

② 设置 URLConnection 的参数和普通请求属性。

③ 如果只是发送 GET 方式请求，使用 connect 方法建立和远程资源之间的实际连接即可；如果需要发送 POST 方式的请求，需要获取 URLConnection 实例对应的输出流来发送请求参数。

④ 远程资源变为可用，程序可以访问远程资源的头字段，或通过输入流读取远程资源的数据。

在建立和远程资源的实际连接之前，程序可以通过如下方法来设置请求头字段。

- setAllowUserInteraction：设置该 URLConnection 的 allowUserInteraction 请求头字段的值。
- setDoInput：设置该 URLConnection 的 doInput 请求头字段的值。
- setDoOutput：设置该 URLConnection 的 doOutput 请求头字段的值。
- setIfModifiedSince：设置该 URLConnection 的 ifModifiedSince 请求头字段的值。
- setUseCaches：设置该 URLConnection 的 useCaches 请求头字段的值。

除此之外，还可以使用如下方法来设置或增加通用头字段。

- setRequestProperty(String key, String value)：设置该 URLConnection 的 key 请求头字段的值为 value。如以下代码所示：

```
conn.setRequestProperty("accept" , "*/*")
```

- addRequestProperty(String key, String value)：为该 URLConnection 的 key 请求头字段的增加 value 值，该方法并不会覆盖原请求头字段的值，而是将新值追加到原请求头字段中。

当远程资源可用之后，程序可以使用以下方法用于访问头字段和内容。

- Object getContent()：获取该 URLConnection 的内容。
- String getHeaderField(String name)：获取指定响应头字段的值。
- getInputStream()：返回该 URLConnection 对应的输入流，用于获取 URLConnection 响应的内容。
- getOutputStream()：返回该 URLConnection 对应的输出流，用于向 URLConnection 发送请求参数。

注意：

如果既要使用输入流读取 URLConnection 响应的内容，也要使用输出流发送请求参数，一定要先使用输出流，再使用输入流。



getHeaderField()方法用于根据响应头字段来返回对应的值。而某些头字段由于经常需要访问，所以 Java 提供了以下方法来访问特定响应头字段的值。

- getContentEncoding：获取 content-encoding 响应头字段的值。
- getContentLength：获取 content-length 响应头字段的值。
- getContentType：获取 content-type 响应头字段的值。
- getDate()：获取 date 响应头字段的值。
- getExpiration()：获取 expires 响应头字段的值。
- getLastModified()：获取 last-modified 响应头字段的值。

下面的程序示范了如何向 Web 站点发送 GET 请求、POST 请求，并从 Web 站点取得响应，该程序中用到一个发送 GET、POST 请求的工具类，该工具类的代码如下。

程序清单：codes\13\13.2\GetPostTest\src\org\crazyit\net\GetPostUtil.java

```
public class GetPostUtil
{
    /**
     * 向指定 URL 发送 GET 方法的请求
     * @param url 发送请求的 URL
     * @param params 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。
```

```
     * @return URL 所代表远程资源的响应
     */
    public static String sendGet(String url, String params)
    {
        String result = "";
        BufferedReader in = null;
        try
        {
            String urlName = url + "?" + params;
            URL realUrl = new URL(urlName);
            // 打开和 URL 之间的连接
            URLConnection conn = realUrl.openConnection();
            // 设置通用的请求属性
            conn.setRequestProperty("accept", "*/*");
            conn.setRequestProperty("connection", "Keep-Alive");
            conn.setRequestProperty("user-agent",
                "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)");
            // 建立实际的连接
            conn.connect(); //①
            // 获取所有响应头字段
            Map<String, List<String>> map = conn.getHeaderFields();
            // 遍历所有的响应头字段
            for (String key : map.keySet())
            {
                System.out.println(key + "--->" + map.get(key));
            }
            // 定义 BufferedReader 输入流来读取 URL 的响应
            in = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
            String line;
            while ((line = in.readLine()) != null)
            {
                result += "\n" + line;
            }
        }
        catch (Exception e)
        {
            System.out.println("发送 GET 请求出现异常！" + e);
            e.printStackTrace();
        }
        // 使用 finally 块来关闭输入流
        finally
        {
            try
            {
                if (in != null)
                {
                    in.close();
                }
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
            }
        }
        return result;
    }
    /**
     * 向指定 URL 发送 POST 方法的请求
     * @param url 发送请求的 URL
     * @param params 请求参数，请求参数应该是 name1=value1&name2=value2 的形式。
```

```
     * @return URL所代表远程资源的响应
     */
    public static String sendPost(String url, String params)
    {
        PrintWriter out = null;
        BufferedReader in = null;
        String result = "";
        try
        {
            URL realUrl = new URL(url);
            // 打开和URL之间的连接
            URLConnection conn = realUrl.openConnection();
            // 设置通用的请求属性
            conn.setRequestProperty("accept", "*/*");
            conn.setRequestProperty("connection", "Keep-Alive");
            conn.setRequestProperty("user-agent",
                "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)");
            // 发送POST请求必须设置如下两行
            conn.setDoOutput(true);
            conn.setDoInput(true);
            // 获取URLConnection对象对应的输出流
            out = new PrintWriter(conn.getOutputStream());
            // 发送请求参数
            out.print(params);  //②
            // flush输出流的缓冲
            out.flush();
            // 定义BufferedReader输入流来读取URL的响应
            in = new BufferedReader(
                new InputStreamReader(conn.getInputStream()));
            String line;
            while ((line = in.readLine()) != null)
            {
                result += "\n" + line;
            }
        }
        catch (Exception e)
        {
            System.out.println("发送POST请求出现异常！" + e);
            e.printStackTrace();
        }
        // 使用finally块来关闭输出流、输入流
        finally
        {
            try
            {
                if (out != null)
                {
                    out.close();
                }
                if (in != null)
                {
                    in.close();
                }
            }
            catch (IOException ex)
            {
                ex.printStackTrace();
            }
        }
        return result;
    }
```

```
}
```

从上面的程序可以看出，如果需要发送 GET 请求，只要调用 URLConnection 的 connect() 方法去建立实际的连接即可，如以上程序中①号粗体字代码所示。如果需要发送 POST 请求，则需要获取 URLConnection 的 OutputStream，然后再向网络中输出请求参数，如以上程序中②号粗体字代码所示。

提供了上面发送 GET 请求、POST 请求的工具类之后，接下来就可以在 Activity 类中通过该工具类来发送请求。该程序的界面中包含两个按钮，一个按钮用于发送 GET 请求，一个按钮用于发送 POST 请求，程序还提供了一个 EditText 来显示远程服务器的响应。该程序的界面布局很简单，故此处不再给出界面布局文件。该程序的 Activity 代码如下。

程序清单：codes\13\13.2\GetPostTest\src\org\crazyit\net\GetPostMain.java

```
public class GetPostMain extends Activity
{
    Button get , post;
    TextView show;
    // 代表服务器响应的字符串
    String response;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            if(msg.what == 0x123)
            {
                // 设置 show 组件显示服务器响应
                show.setText(response);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        get = (Button) findViewById(R.id.get);
        post = (Button) findViewById(R.id.post);
        show = (TextView)findViewById(R.id.show);
        get.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        response = GetPostUtil.sendGet(
                            "http://192.168.1.88:8888/abc/a.jsp"
                            , null);
                        // 发送消息通知 UI 线程更新 UI 组件
                        handler.sendEmptyMessage(0x123);
                    }
                }.start();
            }
        });
        post.setOnClickListener(new OnClickListener()
```

```
        {
            @Override
            public void onClick(View v)
            {
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        response = GetPostUtil.sendPost(
                            "http://192.168.1.88:8888/abc/login.jsp"
                            , "name=crazyit.org&pass=leegang");
                    }
                }.start();
                // 发送消息通知 UI 线程更新 UI 组件
                handler.sendEmptyMessage(0x123);
            }
        });
    }
}
```

上面的程序中两行粗体字代码分别用于发送 GET 请求、POST 请求，该程序所发送的 GET 请求、POST 请求都是向本地局域网内 http://192.168.1.88:8888/abc 应用下两个页面发送，这个应用实际上是部署在笔者本机的 Web 应用。



提示：

abc 这个 Web 应用的代码在光盘的 codes\13\13.2 路径下，这个 Web 应用需要部署在 Web 服务器（比如 Tomcat）中才可使用，关于如何开发 Web 应用，如何安装、部署 Web 应用可参考疯狂 Java 体系的《轻量级 Java EE 企业应用实战》一书。

在 Web 服务器中成功部署 abc 应用之后，运行上面的 Android 应用，单击“发送 GET 请求”按钮将可以看到如图 13.5 所示的输出。

如果单击“发送 POST 请求”按钮，程序将会向 abc 应用下的 login.jsp 页面发送请求，并提交 name=crazyit.org&pass=leegang 请求参数，此时将可以看到如图 13.6 所示的输出。





图 13.5 使用 URLConnection 对外发送 GET 请求　图 13.6 使用 URLConnection 对外发送 POST 请求

从上面的介绍可以发现，借助于 URLConnection 类的帮助，应用程序可以非常方便地与指定站点交换信息：包括发送 GET 请求、POST 请求，并获取网站的响应等。

13.3 使用 HTTP 访问网络

前面介绍了 URLConnection 已经可以非常方便地与指定站点交换信息，URLConnection

还有一个子类：HttpURLConnection，HttpURLConnection 在 URLConnection 的基础上做了进一步改进，增加了一些用于操作 HTTP 资源的便捷方法。

13.3.1 使用 HttpURLConnection

HttpURLConnection 继承了 URLConnection，因此也可用于向指定网站发送 GET 请求、POST 请求。它在 URLConnection 的基础上提供了如下便捷的方法。

- int getResponseCode()：获取服务器的响应代码。
- String getResponseMessage()：获取服务器的响应消息。
- String getRequestMethod()：获取发送请求的方法。
- void setRequestMethod(String method)：设置发送请求的方法。

下面通过一个实用的示例来示范使用 HttpURLConnection 实现多线程下载。

实例：多线程下载

使用多线程下载文件可以更快地完成文件的下载，因为客户端启动多个线程进行下载就意味着服务器也需要为该客户端提供相应的服务。假设服务器同时最多服务 100 个用户，在服务器中一条线程对应一个用户，100 条线程在计算机内并发执行，也就是由 CPU 划分时间片轮流执行，如果 A 应用使用了 99 条线程下载文件，那么相当于占用了 99 个用户的资源，自然就拥有了较快的下载速度。

注意：

实际上并不是客户端并发的下载线程越多，程序的下载速度就越快，因为当客户端开启太多的并发线程之后，应用程序需要维护每条线程的开销、线程同步的开销，这些开销反而会导致下载速度降低。



为了实现多线程下载，程序可按如下步骤进行。

1. 创建 URL 对象。
2. 获取指定 URL 对象所指向资源的大小（由 getContentLength()方法实现），此处用到了 HttpURLConnection 类。
3. 在本地磁盘上创建一个与网络资源相同大小的空文件。
4. 计算每条线程应该下载网络资源的哪个部分（从哪个字节开始，到哪个字节结束）。
5. 依次创建、启动多条线程来下载网络资源的指定部分。

该程序提供的下载工具类代码如下。

程序清单：codes\13\13.3\MultiThreadDown\src\org\crazyit\net\DownUtil.java

```
public class DownUtil
{
    // 定义下载资源的路径
    private String path;
    // 指定所下载的文件的保存位置
    private String targetFile;
    // 定义需要使用多少线程下载资源
    private int threadNum;
    // 定义下载的线程对象
    private DownThread[] threads;
```

```
// 定义下载的文件的总大小
private int fileSize;
public DownUtil(String path, String targetFile, int threadNum)
{
      this.path = path;
      this.threadNum = threadNum;
      // 初始化 threads 数组
      threads = new DownThread[threadNum];
      this.targetFile = targetFile;
}
public void download() throws Exception
{
      URL url = new URL(path);
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      conn.setConnectTimeout(5 * 1000);
      conn.setRequestMethod("GET");
      conn.setRequestProperty(
            "Accept",
            "image/gif, image/jpeg, image/pjpeg, image/pjpeg, "
            + "application/x-shockwave-flash, application/xaml+xml, "
            + "application/vnd.ms-xpsdocument, application/x-ms-xbap, "
            + "application/x-ms-application, application/vnd.ms-excel, "
            + "application/vnd.ms-powerpoint, application/msword, */*");
      conn.setRequestProperty("Accept-Language", "zh-CN");
      conn.setRequestProperty("Charset", "UTF-8");
      conn.setRequestProperty("Connection", "Keep-Alive");
      // 得到文件大小
      fileSize = conn.getContentLength();
      conn.disconnect();
      int currentPartSize = fileSize / threadNum + 1;
      RandomAccessFile file = new RandomAccessFile(targetFile, "rw");
      // 设置本地文件的大小
      file.setLength(fileSize);
      file.close();
      for (int i = 0; i < threadNum; i++)
      {
            // 计算每条线程的下载的开始位置
            int startPos = i * currentPartSize;
            // 每个线程使用一个 RandomAccessFile 进行下载
            RandomAccessFile currentPart = new RandomAccessFile(targetFile,
                  "rw");
            // 定位该线程的下载位置
            currentPart.seek(startPos);
            // 创建下载线程
            threads[i] = new DownThread(startPos, currentPartSize,
                  currentPart);
            // 启动下载线程
            threads[i].start();
      }
}
// 获取下载的完成百分比
public double getCompleteRate()
{
      // 统计多条线程已经下载的总大小
      int sumSize = 0;
      for (int i = 0; i < threadNum; i++)
      {
            sumSize += threads[i].length;
      }
      // 返回已经完成的百分比
      return sumSize * 1.0 / fileSize;
```

```
    }
    private class DownThread extends Thread
    {
        // 当前线程的下载位置
        private int startPos;
        // 定义当前线程负责下载的文件大小
        private int currentPartSize;
        // 当前线程需要下载的文件块
        private RandomAccessFile currentPart;
        // 定义已经该线程已下载的字节数
        public int length;
        public DownThread(int startPos, int currentPartSize,
            RandomAccessFile currentPart)
        {
            this.startPos = startPos;
            this.currentPartSize = currentPartSize;
            this.currentPart = currentPart;
        }
        @Override
        public void run()
        {
            try
            {
                URL url = new URL(path);
                HttpURLConnection conn = (HttpURLConnection)url
                    .openConnection();
                conn.setConnectTimeout(5 * 1000);
                conn.setRequestMethod("GET");
                conn.setRequestProperty(
                    "Accept",
                    "image/gif, image/jpeg, image/pjpeg, image/pjpeg, "
                    + "application/x-shockwave-flash, application/xaml+xml, "
                    + "application/vnd.ms-xpsdocument, application/x-ms-xbap, "
                    + "application/x-ms-application, application/vnd.ms-excel, "
                    + "application/vnd.ms-powerpoint, application/msword, */*");
                conn.setRequestProperty("Accept-Language", "zh-CN");
                conn.setRequestProperty("Charset", "UTF-8");
                InputStream inStream = conn.getInputStream();
                // 跳过 startPos 个字节，表明该线程只下载自己负责哪部分文件
                inStream.skip(this.startPos);
                byte[] buffer = new byte[1024];
                int hasRead = 0;
                // 读取网络数据，并写入本地文件
                while (length < currentPartSize
                    && (hasRead = inStream.read(buffer)) > 0)
                {
                    currentPart.write(buffer, 0, hasRead);
                    // 累计该线程下载的总大小
                    length += hasRead;
                }
                currentPart.close();
                inStream.close();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

上面的 DownUtil 工具类中包括一个 DownloadThread 内部类，该内部类的 run()方法中负责打开远程资源的输入流，并调用 InputStream 的 skip(int)方法跳过指定数量的字节，这样就让该线程读取由它自己负责下载的部分。



提示： 可能有读者会发现，上面这个 DownUtil 类与《疯狂 Java 讲义》中介绍的多线程下载工具类基本相似。实际上也是这样，因此这个类并未用到任何与 Android 相关的知识，其本质就是使用 JDK 提供的 HttpURLConnection 类。



备注： 需要说明的是，上面这个 DownUtil 类在 Java SE 环境下可以正常使用、下载文件，在本书第 1 版的 Android 2.3.3 平台上也可以正常使用、下载文件，在目前在最新的 Android 4.2 模拟器上使用该工具类下载时导致下载的文件有问题（在 Android 4.1 真机上测试完全正常），这是因为在 Android 4.2 模拟器上 InputStream 的 skip(int byteCount)跳过指定字节时，该方法实际上并没有跳过 byteCount 的字节，这可能是 Android 4.2 升级过程中导致的问题。

提供了上面的 DownUtil 工具类之后，接下来就可以在 Activity 中调用该 DownUtil 类来执行下载任务了，该程序界面中包含两个文本框，一个用于输入网络文件的源路径，另一个用于指定下载到本地的文件的文件名，该程序的界面比较简单，故此处不再给出界面布局代码。该程序的 Activity 代码如下。

程序清单：codes\13\13.3\MultiThreadDown\src\org\crazyit\net\MultiThreadDown.java

```
public class MultiThreadDown extends Activity
{
    EditText url;
    EditText target;
    Button downBn;
    ProgressBar bar;
    DownUtil downUtil;
    private int mDownStatus;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面中的三个界面控件
        url = (EditText) findViewById(R.id.url);
        target = (EditText) findViewById(R.id.target);
        downBn = (Button) findViewById(R.id.down);
        bar = (ProgressBar) findViewById(R.id.bar);
        // 创建一个 Handler 对象
        final Handler handler = new Handler()
        {
            @Override
            public void handleMessage(Message msg)
            {
                if (msg.what == 0x123)
                {
                    bar.setProgress(mDownStatus);
                }
```

```
            }
        };
        downBn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 初始化 DownUtil 对象（最后一个参数指定线程数）
                downUtil = new DownUtil(url.getText().toString(),
                    target.getText().toString(), 6);
                new Thread()
                {
                    @Override
                    public void run()
                    {
                        try
                        {
                            // 开始下载
                            downUtil.download();
                        }
                        catch (Exception e)
                        {
                            e.printStackTrace();
                        }
                        // 定义每秒调度获取一次系统的完成进度
                        final Timer timer = new Timer();
                        timer.schedule(new TimerTask()
                        {
                            @Override
                            public void run()
                            {
                                // 获取下载任务的完成比例
                                double completeRate = downUtil.get-
                                CompleteRate();
                                mDownStatus = (int) (completeRate * 100);
                                // 发送消息通知界面更新进度条
                                handler.sendEmptyMessage(0x123);
                                // 下载完全后取消任务调度
                                if (mDownStatus >= 100)
                                {
                                    timer.cancel();
                                }
                            }
                        }, 0, 100);
                    }
                }.start();
            }
        });
    }
}
```

上面的 Activity 不仅使用了 DownUtil 来控制程序下载，而且程序还启动了一个定时器，该定时器控制每隔 0.1 秒查询一次下载进度，并通过程序中的进度条来显示任务的下载进度。

该程序不仅需要访问网络，还需要访问系统 SD 卡，在 SD 卡中创建文件，因此必须授予该程序访问网络、访问 SD 卡文件的权限，也就是在 AndroidManifest.xml 文件中增加如下配置：

```
<!-- 在 SD 卡中创建与删除文件权限 -->
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<!-- 向 SD 卡写入数据权限 -->
```

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!-- 授权访问网络 -->
<uses-permission android:name="android.permission.INTERNET"/>
```

运行该程序将看到如图 13.7 所示的下载界面。



图 13.7 多线程下载



提示：

上面的程序已经实现了多线程下载的核心代码，如果要实现断点下载，则还需要额外增加一个配置文件（读者可以发现所有断点下载工具都会在下载开始生成两个文件：一个是与网络资源相同大小的空文件，一个是配置文件），该配置文件分别记录每个线程已经下载到了哪个字节，当网络断开后再次开始下载时，每个线程根据配置文件里记录的位置向后下载即可。

13.3.2 使用 Apache HttpClient

在一般情况下，如果只是需要向 Web 站点的某个简单页面提交请求并获取服务器响应，完全可以使用前面所介绍的 HttpURLConnection 来完成。但在绝大部分情况下，Web 站点的网页可能没这么简单，这些页面并不是通过一个简单的 URL 就可访问的，可能需要用户登录而且具有相应的权限才可访问该页面。在这种情况下，就需要涉及 Session、Cookie 的处理了，如果打算使用 HttpURLConnection 来处理这些细节，当然也是可能实现的，只是处理起来难度就大了。

为了更好地处理向 Web 站点请求，包括处理 Session、Cookie 等细节问题，Apache 开源组织提供了一个 HttpClient 项目，看它的名称就知道，它是一个简单的 HTTP 客户端（并不是浏览器），可以用于发送 HTTP 请求，接收 HTTP 响应。但不会缓存服务器的响应，不能执行 HTML 页面中嵌入的 JavaScript 代码；也不会对页面内容进行任何解析、处理。



提示：

简单来说，HttpClient 就是一个增强版的 HttpURLConnection，HttpURLConnection 可以做的事情 HttpClient 全部可以做；HttpURLConnection 没有提供的有些功能，HttpClient 也提供了，但它只是关注于如何发送请求、接收响应，以及管理 HTTP 连接。

Android 已经成功地集成了 HttpClient，这意味着开发人员可以直接在 Android 应用中使

用 HttpClient 来访问提交请求、接收响应。

使用 HttpClient 发送请求、接收响应很简单，只要如下几步即可。

1 创建 HttpClient 对象。

2 如果需要发送 GET 请求，创建 HttpGet 对象；如果需要发送 POST 请求，创建 HttpPost 对象。

3 如果需要发送请求参数，可调用 HttpGet、HttpPost 共同的 setParams(HttpParams params)方法来添加请求参数；对于 HttpPost 对象而言，也可调用 setEntity(HttpEntity entity)方法来设置请求参数。

4 调用 HttpClient 对象的 execute(HttpUriRequest request)发送请求，执行该方法返回一个 HttpResponse。

5 调用 HttpResponse 的 getAllHeaders()、getHeaders(String name)等方法可获取服务器的响应头；调用 HttpResponse 的 getEntity()方法可获取 HttpEntity 对象，该对象包装了服务器的响应内容。程序可通过该对象获取服务器的响应内容。

实例：访问被保护资源

下面的 Android 应用需要向指定页面发送请求，但该页面并不是一个简单的页面，只有当用户已经登录，而且登录用户的用户名是 crazyit.org 时才可访问该页面。如果使用 HttpUrlConnection 来访问该页面，那么需要处理的细节就太复杂了。下面将会借助于 HttpClient 来访问被保护的页面。

访问 Web 应用中被保护的页面，如果使用浏览器则十分简单，用户通过系统提供的登录页面登录系统，浏览器会负责维护与服务器之间的 Session，如果用户登录的用户名、密码符合要求，就可以访问被保护资源了。

为了通过 HttpClient 来访问被保护页面，程序同样需要使用 HttpClient 来登录系统，只要应用程序使用同一个 HttpClient 发送请求，HttpClient 会自动维护与服务器之间的 Session 状态，也就是说程序第一次使用 HttpClient 登录系统后，接下来使用 HttpClient 即可访问被保护页面了。



提示：

虽然此处给出的实例只是访问被保护的页面，但访问其他被保护的资源也与此类似。程序只要第一次通过 HttpClient 登录系统，接下来即可通过该 HttpClient 访问被保护资源了。

下面是该程序的代码。

程序清单：codes\13\13.3\HttpClientTest\src\org\crazyit\net\HttpClientTest.java

```
public class HttpClientTest extends Activity
{
    TextView response;
    HttpClient httpClient;
    Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
            if(msg.what == 0x123)
            {
                // 使用 response 文本框显示服务器响应
```

```
                response.append(msg.obj.toString() + "\n");
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 创建 DefaultHttpClient 对象
        httpClient = new DefaultHttpClient();
        response = (TextView) findViewById(R.id.response);
    }
    public void accessSecret(View v)
    {
        response.setText("");
        new Thread()
        {
            @Override
            public void run()
            {
                // 创建一个 HttpGet 对象
                HttpGet get = new HttpGet(
                    "http://192.168.1.88:8888/foo/secret.jsp");  //①
                try
                {
                    // 发送 GET 请求
                    HttpResponse httpResponse = httpClient.execute(get);//②
                    HttpEntity entity = httpResponse.getEntity();
                    if (entity != null)
                    {
                        // 读取服务器响应
                        BufferedReader br = new BufferedReader(
                            new InputStreamReader(entity.getContent()));
                        String line = null;

                        while ((line = br.readLine()) != null)
                        {
                            Message msg = new Message();
                            msg.what = 0x123;
                            msg.obj = line;
                            handler.sendMessage(msg);
                        }
                    }
                }
                catch (Exception e)
                {
                    e.printStackTrace();
                }
            }
        }.start();
    }
    public void showLogin(View v)
    {
        // 加载登录界面
        final View loginDialog = getLayoutInflater().inflate(
            R.layout.login, null);
        // 使用对话框供用户登录系统
        new AlertDialog.Builder(HttpClientTest.this)
            .setTitle("登录系统")
            .setView(loginDialog)
```

```
                .setPositiveButton("登录",
                new DialogInterface.OnClickListener()
                {
                    @Override
                    public void onClick(DialogInterface dialog,
                            int which)
                    {
                        // 获取用户输入的用户名、密码
                        final String name = ((EditText) loginDialog
                                .findViewById(R.id.name)).getText()
                                .toString();
                        final String pass = ((EditText) loginDialog
                                .findViewById(R.id.pass)).getText()
                                .toString();
                        new Thread()
                        {
                            @Override
                            public void run()
                            {
                                try
                                {
                                    HttpPost post = new HttpPost("http://192.168"
                                            + ".1.88:8888/foo/login.jsp");//③
                                    // 如果传递参数个数比较多，可以对传递的参数进行封装
                                    List<NameValuePair> params = new
                                            ArrayList<NameValuePair>();
                                    params.add(new BasicNameValuePair
                                            ("name", name));
                                    params.add(new BasicNameValuePair
                                            ("pass", pass));
                                    // 设置请求参数
                                    post.setEntity(new UrlEncodedFormEntity(
                                            params, HTTP.UTF_8));
                                    // 发送 POST 请求
                                    HttpResponse response = httpClient
                                            .execute(post); //④
                                    // 如果服务器成功地返回响应
                                    if (response.getStatusLine()
                                            .getStatusCode() == 200)
                                    {
                                        String msg = EntityUtils
                                            .toString(response.getEntity());
                                        Looper.prepare();
                                        // 提示登录成功
                                        Toast.makeText(HttpClientTest.this,
                                            msg, Toast.LENGTH_SHORT).show();
                                        Looper.loop();
                                    }
                                }
                                catch (Exception e)
                                {
                                    e.printStackTrace();
                                }
                            }
                        }.start();
                    }
                }).setNegativeButton("取消", null).show();
            }
        }
```

上面的程序中①、②号粗体字代码先创建了一个HttpGet对象，接下来程序调用HttpClient

的 execute()方法发送 GET 请求；程序中③、④号粗体字代码先创建了一个 HttpPost 对象，接下来程序调用了 HttpClient 的 execute()方法发送 POST 请求。上面的 GET 请求用于获取服务器上的被保护页面，POST 请求用于登录系统。

运行该程序，单击“访问页面”按钮将可看到如图 13.8 所示的页面。



提示： 运行该程序需要 Web 应用的支持，读者应该先将 codes/13/13.3 目录下的 foo 应用部署到 Web 服务器（如 Tomcat 7.0）中，然后再运行该应用。

从图 13.8 可以看出，程序直接向指定 Web 应用的被保护页面 secret.jsp 发送请求，程序将无法访问被保护页面，于是看到图 13.8 所示的页面。单击图 13.8 所示页面中的“登录系统”按钮，系统将会显示如图 13.9 所示的登录对话框。

在图 13.9 所示对话框的两个输入框中分别输入“crazyit.org”、“leegang”，然后单击“登录”按钮，系统将会向 Web 站点的 login.jsp 页面发送 POST 请求，并将用户输入的用户名、密码作为请求参数。如果用户名、密码正确，即可看到登录成功的提示。

登录成功后，HttpClient 将会自动维护与服务器之间的连接，并维护与服务器之间的 Session 状态，再次单击程序中的“访问页面”按钮，即可看到如图 13.10 所示的输出。



图 13.8 访问被保护页面



图 13.9 登录对话框



图 13.10 访问被保护资源

从图 13.10 可以看出，此时使用 HttpClient 发送 GET 请求即可正常访问被保护资源，这就是因为前面使用了 HttpClient 登录了系统，而且 HttpClient 可以维护与服务器之间的 Session 连接。

从上面的编程过程不难看出，使用 Apache 的 HttpClient 更加简单，而且它比 HttpURLConnection 提供了更多的功能。

13.4 使用 WebView 视图显示网页

Android 提供了 WebView 组件，表面上来看，这个组件与普通 ImageView 差不多，但实际上这个组件的功能要强大得多，WebView 组件本身就是一个浏览器实现，它的内核基于开源 WebKit 引擎。如果我们对 WebView 进行一些美化、包装，可以非常轻松地开发出自己的浏览器。

13.4.1 使用 WebView 浏览网页

WebView 的用法与普通 ImageView 组件的用法基本相似，它提供了大量方法来执行浏览器操作，例如如下常用方法。

- void goBack()：后退。
- void goForward()：前进。
- void loadUrl(String url)：加载指定 URL 对应的网页。
- boolean zoomIn()：放大网页。
- boolean zoomOut()：缩小网页。

当然 WebView 组件还包含了大量方法，具体以 Android API 文档为准。

下面的程序将基于 WebView 来开发一个简单的浏览器。

实例：迷你浏览器

该程序的界面中包含两个组件：一个文本框用于接收用户输入想访问的 URL；一个 WebView 用于加载并显示该 URL 对应的页面。该程序的界面布局代码如下。

程序清单：codes\13\13.4\MiniBrowser\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
<EditText
      android:id="@+id/url"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      />
<!-- 显示页面的 WebView 组件 -->
<WebView
      android:id="@+id/show"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      />
</LinearLayout>
```

以下程序则主要通过 WebView 的 loadUrl(String url)来加载、显示指定 URL 对应的页面。

程序清单：codes\13\13.4\MiniBrowser\src\org\crazyit\net\MiniBrowser.java

```
public class MiniBrowser extends Activity
{
      EditText url;
      WebView show;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取页面中文本框、WebView 组件
            url = (EditText) findViewById(R.id.url);
            show = (WebView) findViewById(R.id.show);
      }
      @Override
```

```
        public boolean onKeyDown(int keyCode, KeyEvent event)
        {
            if (keyCode == KeyEvent.KEYCODE_SEARCH)
            {
                String urlStr = url.getText().toString();
                // 加载、并显示 urlStr 对应的网页
                show.loadUrl(urlStr);
                return true;
            }
            return false;
        }
    }
```

上面的程序中粗体字代码是该程序的关键，程序调用 WebView 的 loadUrl(String url)方法加载、显示该 URL 对应的网页——至于该 WebView 如何发送请求，如何解析服务器响应，这些细节对用户来说是透明的。

运行该程序，在文本框中输入想访问的站点，并单击手机的“搜索”按钮，将可以看到如图 13.11 所示的输出。

正如图 13.11 所看到的，使用 WebView 开发浏览器十分简单，如果读者愿意多花时间对该程序界面进行美化，并为程序提供前进、后退、刷新等按钮，即可开发出一个实用的浏览器来代替 Android 系统自带的浏览器。

Android 系统自带的浏览器其实也是基于开源的 WebKit 引擎实现的。



图 13.11 使用 WebView 浏览指定网页

13.4.2 使用 WebView 加载 HTML 代码

前面看到使用 EditText 显示 HTML 字符串时十分别扭，EditText 不会对 HTML 标签进行任何解析，而是直接把所有 HTML 标签都显示出来——就像用普通记事本显示一样；如果应用程序想重新对 HTML 字符串进行解析、当成 HTML 页面来显示，也是可以的。

WebView 提供了一个 loadData(String data, String mimeType, String encoding)方法，该方法可用于加载并显示 HTML 代码。但在实际使用过程中，笔者发现这个方法有一个小问题：当它加载包含中文的 HTML 内容时，WebView 将会显示乱码。

好在 WebView 还提供了一个 loadDataWithBaseURL(String baseUrl, String data, String mimeType, String encoding, String historyUrl)方法，该方法是 loadData(String data, String mimeType, String encoding)方法的增强版，它不会产生乱码。关于该方法的几个参数简单说明一下。

- data：指定需要加载的 HTML 代码。
- mimeType：指定 HTML 代码的 MIME 类型，对于 HTML 代码可指定为 text/html。
- encoding：指定 HTML 代码编码所用的字符集。比如指定为 GBK。

下面的程序简单示范了如何使用 WebView 来加载 HTML 代码。

程序清单：codes\13\13.4\ViewHtml\src\org\crazyit\net\ViewHtml.java

```
public class ViewHtml extends Activity
{
    WebView show;
    @Override
    public void onCreate(Bundle savedInstanceState)
```

```
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序中的 WebView 组件
        show = (WebView) findViewById(R.id.show);
        StringBuilder sb = new StringBuilder();
        // 拼接一段 HTML 代码
        sb.append("<html>");
        sb.append("<head>");
        sb.append("<title> 欢迎您 </title>");
        sb.append("</head>");
        sb.append("<body>");
        sb.append("<h2> 欢迎您访问<a href=\"http://www.crazyit.org\">"
            + "疯狂 Java 联盟</a></h2>");
        sb.append("</body>");
        sb.append("</html>");
        // 使用简单的 loadData 方法会导致乱码，可能是 Android API 的 Bug
        // show.loadData(sb.toString() , "text/html" , "utf-8");
        // 加载、并显示 HTML 代码
        show.loadDataWithBaseURL(null, sb.toString()
            , "text/html" , "utf-8", null);
    }
}
```

上面的程序中粗体字代码就是该程序的关键，这行代码负责加载指定 HTML 页面，并将它显示出来，运行该程序将可以看到如图 13.12 所示的输出。

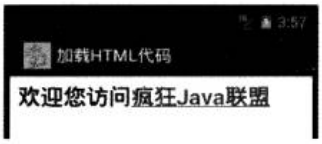


图 13.12　使用 WebView 加载、显示 HTML 代码

13.4.3　使用 WebView 中的 JavaScript 调用 Android 方法

很多时候，WebView 加载的页面上是带 JavaScript 脚本的，比如页面上有一个按钮，用户单击按钮时将会弹出一个提示框，或打开一个列表框等。由于该按钮是 HTML 页面上的按钮，它只能激发一段 JavaScript 脚本，这就需要让 JavaScript 脚本来调用 Android 方法了。

为了让 Web View 中的 JavaScript 脚本调用 Android 方法，WebView 提供了一个配套的 WebSettings 工具类，该工具类提供了大量方法来管理 WebView 的选项设置，其中它的 setJavaScriptEnabled(true)即可让 WebView 中的 JavaScript 脚本来调用 Android 方法。除此之外，为了把 Android 对象暴露给 WebView 中的 JavaScript 代码，WebView 提供了 addJavascriptInterface(Object object, String name)方法，该方法负责把 object 对象暴露成 JavaScript 中的 name 对象。

从上面的介绍可以看出，在 WebView 的 JavaScript 中调用 Android 方法只要如下三个步骤：

① 调用 WebView 关联的 WebSettings 的 setJavaScriptEnabled(true)启用 JavaScript 调用功能。

② 调用 WebView 的 addJavascriptInterface(Object object, String name)方法将 object 对象暴露给 JavaScript。

③ 在 JavaScript 脚本中通过刚才暴露的 name 对象调用 Android 的方法。



提示：

如果读者有 DWR 开发经验，应该很好理解 Android 此处的设计，DWR 通过使用配置，可以让服务端的 Java 对象暴露给 JavaScript 脚本；Android 则通过 WebView 的 addJavascriptInterface()方法把 Android 应用中的对象暴露给 JavaScript 脚本——最后实现的效果是相同的：JavaScript 脚本可以直接调用 Java 对象的方法。

下面的示例示范了如何在 JavaScript 中调用 Android 方法，该示例的界面布局很简单，它包含了一个普通的 WebView 组件，用于显示 HTML 页面。该示例的 Activity 代码如下。

程序清单：codes\13\13.4\JsCallAndroid\src\org\crazyit\net\JsCallAndroid.java

```
public class JsCallAndroid extends Activity
{
    WebView myWebView;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        myWebView = (WebView) findViewById(R.id.webview);
        // 此处为了简化编程，使用 file 协议加载本地 assets 目录下的 HTML 页面
        // 如果有需要，也可使用 http 协议加载远程网站的 HTML 页面
        myWebView.loadUrl("file:///android_asset/test.html");
        // 获取 WebView 的设置对象
        WebSettings webSettings = myWebView.getSettings();
        // 开启 JavaScript 调用
        webSettings.setJavaScriptEnabled(true);
        // 将 MyObject 对象暴露给 JavaScript 脚本
        // 这样 test.html 页面中的 JavaScript 可以通过 myObj 来调用 MyObject 的方法
        myWebView.addJavascriptInterface(new MyObject(this), "myObj");
    }
}
```

上面的程序中第一行粗体字代码开启了 JavaScript 调用 Android 方法的功能，第二行粗体字代码则负责将 Androd 应用中的 MyObject 对象暴露给 JavaScript 脚本，暴露成 JavaScript 脚本中名为 myObj 的对象。

MyObject 是一个自定义的 Java 类，开发者可以根据业务需要提供任意多的方法，本示例只为 MyObject 定义了两个方法。下面是 MyObject 类的代码。

程序清单：codes\13\13.4\JsCallAndroid\src\org\crazyit\net\MyObject.java

```
public class MyObject
{
    Context mContext;
    MyObject(Context c)
    {
        mContext = c;
    }
    // 该方法将会暴露给 JavaScript 脚本调用
    public void showToast(String name)
    {
        Toast.makeText(mContext, name + ", 您好！"
            , Toast.LENGTH_LONG).show();
    }
    // 该方法将会暴露给 JavaScript 脚本调用
    public void showList()
    {
```

```
            // 显示一个普通的列表对话框
            new AlertDialog.Builder(mContext)
                  .setTitle("图书列表")
                  .setIcon(R.drawable.ic_launcher)
                  .setItems(new String[]{"疯狂 Java 讲义"
                  , "疯狂 Android 讲义" , "轻量级 Java EE 企业应用实战"} , null)
                  .setPositiveButton("确定", null)
                  .create()
                  .show();
      }
}
```

正如上面的代码所示，MyObject 中包含了两个方法——showToast()和 showList()方法，这两个方法将会暴露给 JavaScript 脚本，从而允许 JavaScript 脚本通过 myObj 来调用这两个方法。下面是 HTML 页面的代码。

程序清单：codes\13\13.4\JsCallAndroid\assets\test.html

```
<!DOCTYPE html>
<html>
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title> Js 调用 Android </title>
</head>
<body>
<!-- 注意此处的 myObj 是 Android 暴露出来的对象 -->
<input type="button" value="打招呼"
      onclick="myObj.showToast('孙悟空');" />
<input type="button" value="图书列表"
      onclick="myObj.showList();" />
</body>
</html>
```

正如上面的两行粗体字代码所示，当用户单击该页面上的两个按钮时，该页面的 JavaScrip 脚本会通过 myObj 调用 Android 方法。运行该实例，单击第一个按钮，可以看到如图 13.13 所示界面：

如果用户单击第二个按钮，该页面的 JavaScript 脚本将会通过 myObj 调用 Android 的 showList()方法，此时将看到如图 13.14 所示对话框。



图 13.13　JS 调用 Android 方法



图 13.14　JS 调用 Android 方法



备注： 在最新 Android 4.2 模拟器上测试该示例时，无法看到图 13.13、图 13.14 所示运行效果，在 Android 4.1.2 模拟器、真机上测试该示例都可以正常运行。

13.5 使用 Web Service 进行网络编程

Android 应用通常都是运行在手机平台上，手机系统的硬件资源是有限的，不管是存储能力还是计算能力都有限，在 Android 系统上开发、运行一些单用户、小型应用是可能的，但对于需要进行大量的数据处理、复杂计算的应用，还是只能部署在远程服务器上，Android 应用将只是充当这些应用的客户端。

为了让 Android 应用与远程服务器之间进行交互，可以借助于 Java 的 RMI 技术，但这要求远程服务器程序必须采用 Java 实现；也可以借助于 CORBA 技术，但这种技术显得过于复杂；除此之外，Web Service 是一种不错的选择。

13.5.1 Web Service 平台概述

Web Service 平台主要涉及的技术有 SOAP（Simple Object Access Protocol，简单对象访问协议），WSDL（Web Service Description Language，Web Service 描述语言），UDDI（Universal Description，Description and Integration，统一描述、发现和整合协议）。

1. SOAP（简单对象访问协议）

SOAP（Simple Object Access Protocol，简单对象访问协议）是一种具有扩展性的 XML 消息协议。SOAP 允许一个应用程序向另一个应用程序发送 XML 消息，SOAP 消息是从 SOAP 发送者传至 SOAP 接收者的单路消息，任何应用程序均可作为发送者或接收者。SOAP 仅定义消息结构和消息处理的协议，与底层的传输协议独立。因此，SOAP 协议能通过 HTTP，JMS 或 SMTP 协议传输。

SOAP 依赖于 XML 文档来构建，一条 SOAP 消息就是一份特定的 XML 文档，SOAP 消息包含如下三个主要元素：

- 必需的<Envelope.../>根元素，SOAP 消息对应的 XML 文档以该元素作为根元素。
- 可选的<Header../>元素，包含 SOAP 消息的头信息。
- 必需的<Body../>元素，包含所有的调用和响应信息。

就目前的 SOAP 消息的结构来看，<Envelope.../>根元素的通常只能包含两个子元素，第一个子元素是可选的<Header../>元素，第二个子元素是必需的<Body../>元素。

图 13.15 显示了 SOAP 消息的基本结构。

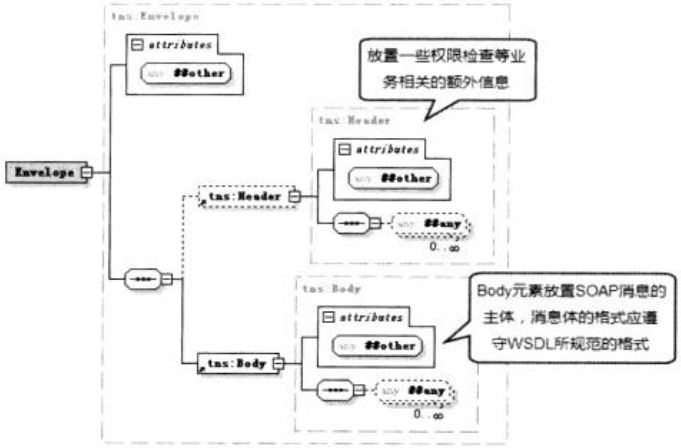


图 13.15 SOAP 消息的基本结构

2. WSDL（Web Service 描述语言）

WSDL（Web Service Description Language，Web Service 描述语言）使用 XML 描述 Web Service，包括访问和使用 Web Service 所必需的信息，定义该 Web Service 的位置、功能及如何通信等描述信息。

一般来说，只要调用者能够获取 Web Service 对应的 WSDL，就可以从中了解它所提供的服务及如何调用 Web Service。因为一份 WSDL 文件清晰地定义了三个方面的内容。

- WHAT 部分：用于定义 Web Service 所提供的操作（或方法），也就是 Web Service 能做些什么。由 WSDL 中的<types.../>、<message.../>和<portType.../>元素定义。
- HOW 部分：用于定义如何访问 Web Service，包括数据格式详情和访问 Web Service 操作的必要协议。也就是定义了如何访问 Web Service。
- WHERE 部分：用于定义 Web Service 位于何处，如何使用特定协议决定的网络地址（如 URL）指定。该部分使用<service.../>元素定义，可在 WSDL 文件的最后部分看到<service.../>元素。

一份 WSDL 文档通常可分为两个部分：

- 第一个部分定义了服务接口，它在 WSDL 中由<message.../>元素和<portType.../>两个元素组成，其中<message.../>元素定义了操作的交互方式。而<portType.../>元素里则可包含任意数量的<operation.../>元素，每个<operation.../>元素代表一个允许远程调用的操作（即方法）。
- WSDL 的第二个部分定义了服务实现，它在 WSDL 中由<binding.../>元素和<service.../>两个元素组成，其中<binding.../>定义使用特定的通信协议、数据编码模型和底层通信协议，将 Web Service 服务接口定义映射到具体实现。而<service.../>元素则包含一系列的<port.../>子元素，<port.../>子元素将会把绑定机制、服务访问协议和端点地址结合在一起。

图 13.16 显示了 WSDL 文档的结构模型。

3. UDDI（统一描述、发现和整合协议）

UDDI（Universal Description，Description and Integration，统一描述、发现和整合协议）是一套信息注册规范，它具有如下特点：

- 基于 Web。
- 分布式。

UDDI 包括一组允许企业向外注册 Web Service、以使其他企业发现访问的实现标准。UDDI 的核心组件是 UDDI 注册中心，它使用 XML 文件来描述企业及其提供的 Web Service。

通过使用 UDDI，Web Service 提供者可以对外注册 Web Service，从而允许其他企业来调用该企业注册的 Web Service。Web Service 提供者通过 UDDI 注册中心的 Web 界面，将它所提供的 Web Service 的信息加入 UDDI 注册中心。该 Web Service 就可以被发现和调用。

Web Service 使用者也通过 UDDI 注册中心查找、发现自己所需的服务。当 Web Service 使用者找到自己所需的服务之后，可以将自己绑定到指定的 Web Service 提供者，再根据该 Web Service 对应的 WSDL 文档来调用对方的服务。

Web Service 大致的运行模式如图 13.17 所示。

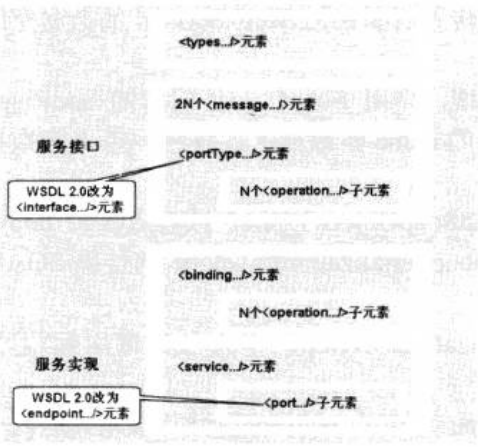


图 13.16 WSDL 文档的结构模型

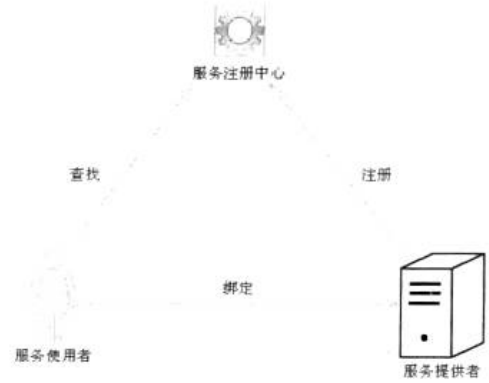


图 13.17 Web Service 示意

13.5.2 使用 Android 应用调用 Web Service

Java 本身提供了丰富的 Web Service 支持，比如 Sun 公司制定的 JAX-WS 2 规范，还有 Apache 开源组织所提供的 Axis1、Axis2、CXF 等，这些技术不仅可以用于非常方便地对外提供 Web Service，也可以用于简化 Web Service 的客户端编程。

对于手机等小型设备而言，它们的计算资源、存储资源都十分有限，因此 Android 应用不大可能需要对外提供 Web Service，Android 应用通常只是充当 Web Service 的客户端，调用远程 Web Serice。

Google 为 Android 平台开发 Web Service 客户端提供了 ksoap2-android 项目，但这个项目并未直接集成在 Android 平台中，还需要开发人员自行下载。

为 Android 应用增加 ksoap2-android 支持请按如下步骤进行。

1. 登录 http://code.google.com/p/ksoap2-android/站点，该站点有介绍下载 ksoap2-android 项目的方法。
2. 下载 ksoap2-android 项目的 ksoap2-android-assembly-3.0.0-RC.4-jar-with-dependencies. jar 包。如果读者下载有困难，也可直接使用光盘的 codes\13\13.5 目录下的该文件。
3. 将下载得到的 JAR 包添加到 Android 项目的 libs 目录下，即可在 Eclipse 左边看到如图 13.18 所示的项目管理树。



图 13.18 为 Android 项目添加 ksoap2-android 包

为 Android 项目添加了 ksoap2-android 包之后，接下来借助于 ksoap2-android 项目来调用 Web Service 所暴露的操作。

使用 ksoap2-android 调用 Web Service 操作的步骤如下：

1. 创建 HttpTransportSE 对象，该对象用于调用 Web Service 操作。
2. 创建 SoapSerializationEnvelope 对象。



提示： 从名称来看，SoapSerializationEnvelope 代表一个 SOAP 消息封包；但 ksoap2-android 项目对 SoapSerializationEnvelope 的处理比较特殊，它是 HttpTransportSE 调用 Web Service 时信息的载体；客户端需要传入的参数，需要通过 SoapSerializationEnvelope 对象的 bodyOut 属性传给服务器；服务器响应生成的 SOAP 消息也通过该对象的 bodyIn 属性来获取。

③ 创建 SoapObject 对象，创建该对象时需要传入所要调用 Web Service 的命名空间、Web Service 方法名。

④ 如果有参数需要传给 Web Service 服务器端，调用 SoapObject 对象的 addProperty (String name , Object value)方法来设置参数，该方法的 name 参数指定参数名；value 参数指定参数值。

⑤ 调用 SoapSerializationEnvelope 的 setOutputSoapObject()方法，或者直接对 bodyOut 属性赋值，将前两步创建的 SoapObject 对象设为 SoapSerializationEnvelope 的传出 SOAP 消息体。

⑥ 调用对象的 call()方法，并以 SoapSerializationEnvelope 作为参数调用远程 Web Service。

⑦ 调用完成后，访问 SoapSerializationEnvelope 对象的 bodyIn 属性，该属性返回一个 SoapObject 对象，该对象就代表了 Web Service 的返回消息。解析该 SoapObject 对象，即可获取调用 Web Service 的返回值。

实例：调用基于 CXF 的 Web Service

下面的程序使用 CXF 开发了一个 Web Service，该 Web Service 对应的 WSDL 文档如图 13.19 所示。



图 13.19　Web Service 提供的 WSDL



提示：

使用 Ant 通过光盘下 codes\13\13.5\WsServer 目录下的 build.xml 文件运行 Web Service 服务器端——当服务器运行起来之后才可通过浏览器看到图 13.19 所示的 WSDL 文档。

正如图 13.19 所看到的，Web Service 客户端只要访问到 Web Service 的 WSDL 文档，即可根据该文本来获取调用 Web Service 操作的必要信息。



提示：

由于本书的重点是介绍 Android 应用调用 Web Service，因此不会涉及如何使用 CXF 开发 Web Service，以及 Web Service 的 WSDL 文档中各元素的作用、意义等知识；如果读者对 Web Service 服务器端的开发有兴趣，可以参考疯狂 Java 体系的《疯狂 XML 讲义》一书。

通过图 13.19 所示的 WSDL 文档了解到调用 Web Service 的方法名、所需的参数名之后，接下来就可以在 Android 程序中通过 ksoap2-android 调用 Web Service 操作。

下面的程序示范了如何通过 ksoap2-android 来调用 Web Service 操作，该程序的界面很简单，界面中只定义了两个文本框来装载服务器响应，因此此处不再给出界面布局代码。该程序的 Activity 代码如下。

程序清单：codes\13\13.5\CallWs\src\org\crazyit\net\CallWs.java

```
public class CallWs extends Activity
{
    final static String SERVICE_NS = "http://lee/";
    final static String SERVICE_URL = "http://192.168.1.88:9999/crazyit";
    private EditText txt1;
    private EditText txt2;
    Handler handler = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            switch(msg.what)
            {
                case 0x123:
                    txt1.setText(msg.obj.toString());
                    break;
                case 0x234:
                    txt2.setText(msg.obj.toString());
                    break;
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        txt1 = (EditText) findViewById(R.id.txt1);
        txt2 = (EditText) findViewById(R.id.txt2);
        // 调用的方法
        String methodName = "getUserList";
        // 创建 HttpTransportSE 传输对象
        final HttpTransportSE ht = new HttpTransportSE(SERVICE_URL); //①
        ht.debug = true;
        // 使用 SOAP1.1 协议创建 Envelop 对象
        final SoapSerializationEnvelope envelope = new
            SoapSerializationEnvelope(SoapEnvelope.VER11); //②
        // 实例化 SoapObject 对象
        SoapObject soapObject = new SoapObject(SERVICE_NS, methodName); //③
        soapObject.addProperty("arg0", "客户端参数:"); //④
        // 将 soapObject 对象设置为 SoapSerializationEnvelope 对象的传出 SOAP 消息
        envelope.bodyOut = soapObject; //⑤
        new Thread()
        {
            public void run()
            {
                try
                {
                    // 调用 Web Service
                    ht.call(null, envelope); //⑥
                    if (envelope.getResponse() != null)
                    {
```

```
                                        // 获取服务器响应返回的 SOAP 消息
                                        SoapObject result = (SoapObject) envelope.bodyIn; //⑦
                                        // 接下来就是从 SoapObject 对象中解析响应数据的过程了
                                        SoapObject detail1 = (SoapObject) result
                                            .getProperty(0);
                                        SoapObject detail2 = (SoapObject) result
                                            .getProperty(1);
                                        StringBuilder person1 = new StringBuilder();
                                        person1.append("用户名：");
                                        person1.append(detail1.getProperty(3));
                                        person1.append("\n 密码");
                                        person1.append(detail1.getProperty(0));
                                        person1.append("\n 身高：");
                                        person1.append(detail1.getProperty(1));
                                        Message msg = new Message();
                                        msg.what = 0x123;
                                        msg.obj = person1.toString();
                                        handler.sendMessage(msg);
                                        StringBuilder person2 = new StringBuilder();
                                        person2.append("用户名：");
                                        person2.append(detail2.getProperty(3));
                                        person2.append("\n 密码：");
                                        person2.append(detail2.getProperty(0));
                                        person2.append("\n 身高：");
                                        person2.append(detail2.getProperty(1));
                                        Message msg2 = new Message();
                                        msg2.what = 0x234;
                                        msg2.obj = person2.toString();
                                        handler.sendMessage(msg2);
                                    }
                                }
                                catch (IOException e)
                                {
                                    e.printStackTrace();
                                }
                                catch (XmlPullParserException e)
                                {
                                    e.printStackTrace();
                                }
                            }
                        }.start();
                    }
                }
```

上面的程序中粗体字代码就代表了使用 ksoap2-android 来调用 Web Service 操作的关键的 7 个步骤。

Web Service 服务器端运行起来之后，运行上面的 Android 应用，即可看到如图 13.20 所示的输出。



图 13.20　调用 Web Service

如图 13.20 所示的两个文本框中看到的内容就是调用 Web Service 所返回的数据。由此可见，不管远程 Web Service 提供的服务功能多么强大、业务实现多么复杂，对于 Android 客户端是完全透明的，Android 只要送出相应的请求参数，服务器就会返回包含结果的 SOAP 消息。借助于 Web Service 这个桥梁，在 Android 应用中实现功能非常强大的应用——反正具体的业务逻辑由 Web Service 服务器端实现，Android 客户端只要调用 Web Service 服务即可。

下面将开发一个实用的案例，这个案例会调用远程 Web Service 来实现天气预报功能。

实例：调用 Web Service 实现天气预报

在开发天气预报的 Android 应用之前，首先需要找到一个可以对外提供天气预报的 Web Service，通过搜索，发现 http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx 站点可以对外提供天气预报的 Web Service，因此程序将会调用该站点的 Web Service 来实现天气预报。



提示：

如果读者学习本程序时，该站点的天气预报的 Web Service 服务已经停止，那么本程序将无法正常调用 Web Service，那么天气预报功能自然也就失效了。

访问 http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx?wsdl 即可看到如图 13.21 所示的 WSDL 文档。



图 13.21 天气预报 Web Service 的 WSDL 文档

通过如图 13.21 所示的 WSDL 文档即可查看到调用 Web Service 的必要信息，如果读者阅读 WSDL 文档存在困难，访问 http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx 站点也可看到调用天气预报 Web Service 的详细说明。

为了让应用界面更加美观，可以访问 http://www.webxml.com.cn/images/weather.zip 下载各种天气图标，可以使用这些天气图标来美化应用。

本程序主要需要调用如下三个 Web Service 操作：

- 获取省份。
- 根据省份获取城市。
- 根据城市获取天气。

为了调用上面的三个 Web Service，应用程序提供如下工具类。

程序清单：codes\13\13.5\MyWeather\src\org\crazyit\net\WebServiceUtil.java

```
public class WebServiceUtil
{
    // 定义 Web Service 的命名空间
    static final String SERVICE_NS = "http://WebXml.com.cn/";
    // 定义 Web Service 提供服务的 URL
    static final String SERVICE_URL =
        "http://webservice.webxml.com.cn/WebServices/WeatherWS.asmx";
    // 调用远程 Web Service 获取省份列表
```

```
public static List<String> getProvinceList()
{
    // 调用的方法
    final String methodName = "getRegionProvince";
    // 创建 HttpTransportSE 传输对象
    final HttpTransportSE ht = new HttpTransportSE(SERVICE_URL);
    ht.debug = true;
    // 使用 SOAP1.1 协议创建 Envelop 对象
    final SoapSerializationEnvelope envelope =
        new SoapSerializationEnvelope(SoapEnvelope.VER11);
    // 实例化 SoapObject 对象
    SoapObject soapObject = new SoapObject(SERVICE_NS, methodName);
    envelope.bodyOut = soapObject;
    // 设置与.NET 提供的 Web Service 保持较好的兼容性
    envelope.dotNet = true;
    FutureTask<List<String>> task = new FutureTask<List<String>>(
    new Callable<List<String>>()
    {
        @Override
        public List<String> call()
            throws Exception
        {
            // 调用 Web Service
            ht.call(SERVICE_NS + methodName, envelope);
            if (envelope.getResponse() != null)
            {
                // 获取服务器响应返回的 SOAP 消息
                SoapObject result = (SoapObject) envelope.bodyIn;
                SoapObject detail = (SoapObject) result.getProperty(
                    methodName + "Result");
                // 解析服务器响应的 SOAP 消息
                return parseProvinceOrCity(detail);
            }
            return null;
        }
    });
    new Thread(task).start();
    try
    {
        return task.get();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    return null;
}
// 根据省份获取城市列表
public static List<String> getCityListByProvince(String province)
{
    // 调用的方法
    final String methodName = "getSupportCityString";
    // 创建 HttpTransportSE 传输对象
    final HttpTransportSE ht = new HttpTransportSE(SERVICE_URL);
    ht.debug = true;
    // 实例化 SoapObject 对象
    SoapObject soapObject = new SoapObject(SERVICE_NS, methodName);
    // 添加一个请求参数
    soapObject.addProperty("theRegionCode", province);
    // 使用 SOAP1.1 协议创建 Envelop 对象
    final SoapSerializationEnvelope envelope =
```

```
            new SoapSerializationEnvelope(SoapEnvelope.VER11);
        envelope.bodyOut = soapObject;
        // 设置与.Net提供的Web Service保持较好的兼容性
        envelope.dotNet = true;
        FutureTask<List<String>> task = new FutureTask<List<String>>(
        new Callable<List<String>>()
        {
            @Override
            public List<String> call()
                throws Exception
            {
                // 调用Web Service
                ht.call(SERVICE_NS + methodName, envelope);
                if (envelope.getResponse() != null)
                {
                    // 获取服务器响应返回的SOAP消息
                    SoapObject result = (SoapObject) envelope.bodyIn;
                    SoapObject detail = (SoapObject) result.getProperty(
                        methodName + "Result");
                    // 解析服务器响应的SOAP消息
                    return parseProvinceOrCity(detail);
                }
                return null;
            }
        });
        new Thread(task).start();
        try
        {
            return task.get();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return null;
    }
    private static List<String> parseProvinceOrCity(SoapObject detail)
    {
        ArrayList<String> result = new ArrayList<String>();
        for (int i = 0; i < detail.getPropertyCount(); i++)
        {
            // 解析出每个省份
            result.add(detail.getProperty(i).toString().split(",")[0]);
        }
        return result;
    }
    public static SoapObject getWeatherByCity(String cityName)
    {
        final String methodName = "getWeather";
        final HttpTransportSE ht = new HttpTransportSE(SERVICE_URL);
        ht.debug = true;
        final SoapSerializationEnvelope envelope =
            new SoapSerializationEnvelope(SoapEnvelope.VER11);
        SoapObject soapObject = new SoapObject(SERVICE_NS, methodName);
        soapObject.addProperty("theCityCode", cityName);
        envelope.bodyOut = soapObject;
        // 设置与.NET提供的Web Service保持较好的兼容性
        envelope.dotNet = true;
        FutureTask<SoapObject> task = new FutureTask<SoapObject>(
        new Callable<SoapObject>()
        {
```

```
                @Override
                public SoapObject call()
                        throws Exception
                {
                        ht.call(SERVICE_NS + methodName, envelope);
                        SoapObject result = (SoapObject) envelope.bodyIn;
                        SoapObject detail = (SoapObject) result.getProperty(
                                methodName + "Result");
                        return detail;
                }
        });
        new Thread(task).start();
        try
        {
                return task.get();
        }
        catch (Exception e)
        {
                e.printStackTrace();
        }
        return null;
    }
}
```

上面的程序调用 Web Service 的方法还是没有改变，前面两个方法——获取系统支持的省份列表，根据省份获取城市列表——将远程 Web Service 返回的数据解析成 List<String>后返回，这样方便 Android 应用使用。由于第二个方法需要返回的数据量较多，所以程序直接返回了 SoapObject 对象。

上面的程序中调用 Web Service 时将 SoapSerializationEnvelope 对象的 dotNet 属性设为 true——因为上面这个网站是通过.NET 来对外提供 Web Service 的，因此需要将 SoapSerializationEnvelope 对象的 dotNet 属性设为 true。

有了上面的调用 Web Service 的工具类之后，接下来可以在 Activity 中使用该工具类来获取天气服务信息。该 Activity 使用了两个 Spinner 让用户选择省份、城市，当用户选择指定城市后，系统将会加载该程序的天气信息。

该程序的界面布局代码如下。

程序清单：codes\13\13.5\MyWeather\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
<LinearLayout
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
<TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:hint="@string/province"/>
<!-- 让用户选择省份的 Spinner -->
<Spinner
      android:id="@+id/province"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"/>
</LinearLayout>
<LinearLayout
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/city"/>
    <!-- 让用户选择城市的 Spinner -->
    <Spinner
        android:id="@+id/city"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    </LinearLayout>
    <!-- 显示今天天气的图片和文本框 -->
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/todayWhIcon1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <ImageView
        android:id="@+id/todayWhIcon2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/weatherToday"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    </LinearLayout>
    <!-- 显示明天天气的图片和文本框 -->
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/tomorrowWhIcon1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <ImageView
        android:id="@+id/tomorrowWhIcon2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/weatherTomorrow"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    </LinearLayout>
    <!-- 显示后天天气的图片和文本框 -->
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/afterdayWhIcon1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <ImageView
        android:id="@+id/afterdayWhIcon2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
```

```
        android:id="@+id/weatherAfterday"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    </LinearLayout>
    <TextView
        android:id="@+id/weatherCurrent"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"/>
    </LinearLayout>
```

当程序加载时，程序会调用 WebServiceUtil 的 getProvinceList()方法来获取省份列表，并使用第一个 Spinner 加载、显示所有省份；当用户改变选择了省份之后，程序会调用 WebServiceUtil 的 getCityListByProvince(String province)方法来获取该省份的全部城市；当用户改变选择城市之后，程序会调用 WebServiceUtil 的 getWeatherByCity(String cityName)方法获取该城市的天气。

该 Activity 的代码如下。

程序清单：codes\13\13.5\MyWeather\src\org\crazyit\net\MyWeather.java

```
public class MyWeather extends Activity
{
    private Spinner provinceSpinner;
    private Spinner citySpinner;
    private ImageView todayWhIcon1;
    private ImageView todayWhIcon2;
    private TextView textWeatherToday;
    private ImageView tomorrowWhIcon1;
    private ImageView tomorrowWhIcon2;
    private TextView textWeatherTomorrow;
    private ImageView afterdayWhIcon1;
    private ImageView afterdayWhIcon2;
    private TextView textWeatherAfterday;
    private TextView textWeatherCurrent;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        todayWhIcon1 = (ImageView) findViewById(R.id.todayWhIcon1);
        todayWhIcon2 = (ImageView) findViewById(R.id.todayWhIcon2);
        textWeatherToday = (TextView) findViewById(R.id.weatherToday);
        tomorrowWhIcon1 = (ImageView) findViewById(R.id.tomorrowWhIcon1);
        tomorrowWhIcon2 = (ImageView) findViewById(R.id.tomorrowWhIcon2);
        textWeatherTomorrow = (TextView) findViewById(R.id.weatherTomorrow);
        afterdayWhIcon1 = (ImageView) findViewById(R.id.afterdayWhIcon1);
        afterdayWhIcon2 = (ImageView) findViewById(R.id.afterdayWhIcon2);
        textWeatherAfterday = (TextView) findViewById(R.id.weatherAfterday);
        textWeatherCurrent = (TextView) findViewById(R.id.weatherCurrent);
        // 获取程序界面中选择省份、城市的 Spinner 组件
        provinceSpinner = (Spinner) findViewById(R.id.province);
        citySpinner = (Spinner) findViewById(R.id.city);
        // 调用远程 Web Service 获取省份列表
        List<String> provinces = WebServiceUtil.getProvinceList();
        ListAdapter adapter = new ListAdapter(this, provinces);
        // 使用 Spinner 显示省份列表
        provinceSpinner.setAdapter(adapter);
        // 当省份 Spinner 的选择项被改变时
        provinceSpinner.setOnItemSelectedListener(new OnItemSelectedListener()
        {
```

```
            @Override
            public void onItemSelected(AdapterView<?> source, View parent,
                  int position, long id)
            {
                  List<String> cities = WebServiceUtil
                        .getCityListByProvince(provinceSpinner.getSelectedItem()
                              .toString());
                  ListAdapter cityAdapter = new ListAdapter(MyWeather.this,
                        cities);
                  // 使用 Spinner 显示城市列表
                  citySpinner.setAdapter(cityAdapter);
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
            }
      });
      // 当城市 Spinner 的选择项被改变时
      citySpinner.setOnItemSelectedListener(new OnItemSelectedListener()
      {
            @Override
            public void onItemSelected(AdapterView<?> source, View parent,
                  int position, long id)
            {
                  showWeather(citySpinner.getSelectedItem().toString());
            }
            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
            }
      });
}
private void showWeather(String city)
{
      String weatherToday = null;
      String weatherTomorrow = null;
      String weatherAfterday = null;
      String weatherCurrent = null;
      int iconToday[] = new int[2];
      int iconTomorrow[] = new int[2];
      int iconAfterday[] = new int[2];
      // 获取远程 Web Service 返回的对象
      SoapObject detail = WebServiceUtil.getWeatherByCity(city);
      // 获取天气实况
      weatherCurrent = detail.getProperty(4).toString();
      // 解析今天的天气情况
      String date = detail.getProperty(7).toString();
      weatherToday = "今天：" + date.split(" ")[0];
      weatherToday = weatherToday + "\n天气：" + date.split(" ")[1];
      weatherToday = weatherToday + "\n气温："
            + detail.getProperty(8).toString();
      weatherToday = weatherToday + "\n风力："
            + detail.getProperty(9).toString() + "\n";
      iconToday[0] = parseIcon(detail.getProperty(10).toString());
      iconToday[1] = parseIcon(detail.getProperty(11).toString());
      // 解析明天的天气情况
      date = detail.getProperty(12).toString();
      weatherTomorrow = "明天：" + date.split(" ")[0];
      weatherTomorrow = weatherTomorrow + "\n天气：" + date.split(" ")[1];
      weatherTomorrow = weatherTomorrow + "\n气温："
            + detail.getProperty(13).toString();
      weatherTomorrow = weatherTomorrow + "\n风力："
```

```
                + detail.getProperty(14).toString() + "\n";
            iconTomorrow[0] = parseIcon(detail.getProperty(15).toString());
            iconTomorrow[1] = parseIcon(detail.getProperty(16).toString());
            // 解析后天的天气情况
            date = detail.getProperty(17).toString();
            weatherAfterday = "后天：" + date.split(" ")[0];
            weatherAfterday = weatherAfterday + "\n天气：" + date.split(" ")[1];
            weatherAfterday = weatherAfterday + "\n气温："
                + detail.getProperty(18).toString();
            weatherAfterday = weatherAfterday + "\n风力："
                + detail.getProperty(19).toString() + "\n";
            iconAfterday[0] = parseIcon(detail.getProperty(20).toString());
            iconAfterday[1] = parseIcon(detail.getProperty(21).toString());
            // 更新当天的天气实况
            textWeatherCurrent.setText(weatherCurrent);
            // 更新显示今天天气的图标和文本框
            textWeatherToday.setText(weatherToday);
            todayWhIcon1.setImageResource(iconToday[0]);
            todayWhIcon2.setImageResource(iconToday[1]);
            // 更新显示明天天气的图标和文本框
            textWeatherTomorrow.setText(weatherTomorrow);
            tomorrowWhIcon1.setImageResource(iconTomorrow[0]);
            tomorrowWhIcon2.setImageResource(iconTomorrow[1]);
            // 更新显示后天天气的图标和文本框
            textWeatherAfterday.setText(weatherAfterday);
            afterdayWhIcon1.setImageResource(iconAfterday[0]);
            afterdayWhIcon2.setImageResource(iconAfterday[1]);
        }
        // 工具方法，该方法负责把返回的天气图标字符串，转换为程序的图片资源 ID
        private int parseIcon(String strIcon)
        {
            // 省略了根据字符串解析天气图标的代码
            ...
        }
}
```

上面的 Activity 代码已经不再涉及调用 Web Service 的代码了，只是简单地调用 Web Service 操作，解析 Web Service 返回的 SOAP 消息包，并把 SOAP 消息包中的数据显示出来。运行该程序，选择指定城市，即可看到如图 13.22 所示的输出。

图 13.22 就是调用 Web Service 开发的天气预报应用，从这个示例应用可以看出，不管远程 Web Service 的功能多么复杂，对于 Android 客户端而言，它只要通过 Web Service 获取对方提供的数据，并将这些数据“整合”到自己的应用中即可，因此 Web Service 为 Android 应用提供了强大后台支持。

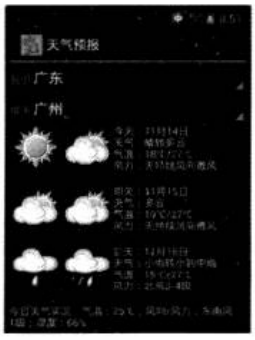


图 13.22　天气预报

13.6　本章小结

本章主要介绍了 Android 的网络编程的相关知识，由于 Android 完全支持 JDK 网络编程中的 ServerSocket、Socket、DatagramSocket、Datagrampacket、MulticastSocket 等 API，也支持 JDK 内置的 URL、URLConnection、HttpURLConnection 等工具类，因此如果读者已经具有网络编程的经验，这些经验完全适用于 Android 网络编程。除此之外，Android 还内置了 Apache HttpClient 支持，通过 HttpClient 可以非常方便地维持与服务器的会话状态、发送请求、获取响应等。虽然 Android 本身没有内置 Web Service 支持，但本章介绍了通过 ksoap2-android 项目调用远程 Web Service 的相关内容，这也是需要读者掌握的内容。

CHAPTER

14

第14章 管理 Android 手机桌面

本章要点

- Android 手机桌面的概念
- 改变手机壁纸的 API
- 动态壁纸的 API
- 开发动态壁纸
- 向 Android 桌面添加快捷方式
- 桌面控件的概念
- 添加桌面控件
- 带数据集的桌面控件
- RemoteViewsService 和 RemoteViewsFactory





Android 系统提供了一个桌面——也就是用户启动后第一次看到的界面，如图 14.1 所示。从图 14.1 可以看出，手机桌面的作用类似于 PC 的桌面，桌面上通常用于放置一些常用的程序和功能。

Android 桌面上首先看到的壁纸，也就是手机桌面上的那张图片，接着可以看到手机桌面规则排列的多个图标，这些图标就是 Android 桌面组件，分别代表快捷方式与桌面控件；每个快捷方式只占用桌面的一个摆放位置；桌面控件则可以很大，一个桌面控件就可以占据多个摆放位置。

Android 系统提供了很好的可扩展性，开发者完全可以在程序中管理 Android 桌面，包括改变系统壁纸、管理快捷方式与创建桌面控件等。



图 14.1　Android 桌面

14.1　管理手机桌面

默认情况下，Android 系统的桌面上除了一张壁纸之外，并不会显示如图 14.1 所示的组件。Android 系统允许普通用户动态地添加、删除桌面组件。

14.1.1　删除桌面组件

默认情况下，Android 系统的桌面上会显示两个桌面组件：相机和模拟时钟，模拟时钟就是搜索条下面的组件。一旦熟悉了 Android 系统之后，用户可能并不希望总是看到这两个组件，此时可能会考虑删除该组件。

删除 Android 桌面组件的步骤如下。

① 在屏幕上长按指定组件，直到桌面上方出现“删除”图标。

② 将指定组件拖到桌面上方的“删除”图标上，如图 14.2 所示。

14.1.2　添加桌面组件

Android 桌面组件可分为快捷方式与桌面控件两种，两种组件的添加方式略有不同。

为了在 Android 桌面上添加快捷方式，可以按如下步骤进行。

① 进入手机的程序列表界面，长按需要添加快捷方式的程序，此时可以看到程序图标会自动切换到桌面，如图 14.3 所示。



图 14.2　删除桌面组件



图 14.3　添加快捷方式

② 在图 14.3 所示界面中拖动该程序图标，将它放到指定位置即可。

为了在 Android 桌面上添加桌面控件，可以按如下步骤进行。

① 进入手机的程序列表界面，然后单击屏幕左上角的“Widget”标签，系统进入如图 14.4 所示的 Widget 列表界面。

② 在图 14.4 所示界面中，长按需要添加的桌面控件，此时可以看到桌面控件会自动切换到桌面，如图 14.5 所示。



图 14.4 桌面控件列表



图 14.5 添加桌面控件

③ 在图 14.5 所示界面中拖动该桌面控件，将它放到指定位置即可。

Android 系统提供的上面的操作能让用户动态地管理手机桌面，但这些能在桌面显示的组件都是 Android 系统本身提供的，接下来将会介绍如何通过应用程序来管理桌面组件。

14.2 改变手机壁纸

前面介绍系统服务时已经介绍了一个“定时更换壁纸”的应用，在那个应用中，当用户启动相应的 Service 之后，该 Service 将会根据 AlarmManager 来定时改变手机壁纸。

Android 允许使用 WallpaperManager 来改变手机壁纸，该对象中改变手机壁纸的方法如下。

- setBitmap(Bitmap bitmap)：将壁纸设置为 bitmap 所代表的位图。
- setResource(int resid)：将壁纸设置为 resid 资源所代表的图片。
- setStream(InputStream data)：将壁纸设置为 data 数据所代表的图片。

前面介绍的这种改变手机壁纸的方式只是动态地切换不同图片作为手机壁纸，此处不再介绍调用 WallpaperManager 来改变手机壁纸的示例。

除此之外，Android 系统还提供了一种动态壁纸的功能，例如用户在手机桌面上长按，系统将会显示如图 14.6 所示的更改壁纸的方式。

如果用户单击图 14.6 所示的“动态壁纸”列表项，系统将会显示 2 个动态壁纸，这是 Android 系统默认提供的两个动态壁纸。除此之外，开发者可以开发任意的动态壁纸。



图 14.6 动态壁纸

14.2.1 开发动态壁纸（Live Wallpapers）

所谓动态壁纸，就是指手机桌面不再是简单的图片，而是运行中的动画，这个动画是由程序实时绘制的，因此被称为动态壁纸。

为了开发动态壁纸，Android 提供了 WallpaperService 基类，动态壁纸的实现类需要继承该基类。在 Android 应用中开发动态壁纸的步骤如下。

① 开发一个子类继承 WallpaperService 基类。

② 继承 WallpaperService 基类时必须重写 onCreateEngine()方法，该方法返回 WallpaperService.Engine 子类对象。

③ 开发者需要实现 WallpaperService.Engine 子类，并重写其中的 public void onVisibilityChanged (boolean visible)、public void onOffsetsChanged()方法。不仅如此，由于 WallpaperService.Engine 子类采用了与 SurfaceView 相同的绘图机制，因此还可选择性地重写在 SurfaceHolder.Callback 中的三个方法。重写这些方法时可通过 SurfaceHolder 动态地绘制图形。

实例：蜿蜒壁纸

下面实例将通过一个不断变幻的矩形在桌面上绘制动态壁纸。下面的 Service 类就代表了动态壁纸服务，程序代码如下。

程序清单：codes\14\14.2\LiveWallPaper\src\org\crazyit\desktop\LiveWallpaper.java

```
public class LiveWallpaper extends WallpaperService
{
    // 记录用户触碰点的位图
    private Bitmap heart;
    // 实现 WallpaperService 必须实现的抽象方法
    @Override
    public Engine onCreateEngine()
    {
        // 加载心形图片
        heart = BitmapFactory.decodeResource(getResources()
            , R.drawable.heart);
        // 返回自定义的 Engine
        return new MyEngine();
    }
    class MyEngine extends Engine
    {
        // 记录程序界面是否可见
        private boolean mVisible;
        // 记录当前当前用户动作事件的发生位置
        private float mTouchX = -1;
        private float mTouchY = -1;
        // 记录当前需要绘制的矩形的数量
        private int count = 1;
        // 记录绘制第一个矩形所需坐标变换的 X、Y 坐标的偏移
        private int originX = 50 , originY = 50;
        // 定义画笔
        private Paint mPaint = new Paint();
        // 定义一个 Handler
        Handler mHandler = new Handler();
        // 定义一个周期性执行的任务
        private final Runnable drawTarget = new Runnable()
        {
            public void run()
            {
                drawFrame();
            }
        };
        @Override
        public void onCreate(SurfaceHolder surfaceHolder)
        {
            super.onCreate(surfaceHolder);
            // 初始化画笔
```

```
        mPaint.setARGB(76 , 0, 0 , 255);
        mPaint.setAntiAlias(true);
        mPaint.setStyle(Paint.Style.FILL);
        // 设置处理触摸事件
        setTouchEventsEnabled(true);
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
        // 删除回调
        mHandler.removeCallbacks(drawTarget);
    }
    @Override
    public void onVisibilityChanged(boolean visible)
    {
        mVisible = visible;
        // 当界面可见时候，执行 drawFrame()方法
        if (visible)
        {
            // 动态地绘制图形
            drawFrame();
        }
        else
        {
            // 如果界面不可见，删除回调
            mHandler.removeCallbacks(drawTarget);
        }
    }
    @Override
    public void onOffsetsChanged(float xOffset, float yOffset, float xStep,
        float yStep, int xPixels, int yPixels)
    {
        drawFrame();
    }
    @Override
    public void onTouchEvent(MotionEvent event)
    {
        // 如果检测到滑动操作
        if (event.getAction() == MotionEvent.ACTION_MOVE)
        {
            mTouchX = event.getX();
            mTouchY = event.getY();
        }
        else
        {
            mTouchX = -1;
            mTouchY = -1;
        }
        super.onTouchEvent(event);
    }
    // 定义绘制图形的工具方法
    private void drawFrame()
    {
    // 获取该壁纸的 SurfaceHolder
        final SurfaceHolder holder = getSurfaceHolder();
        Canvas c = null;
        try
        {
            // 对画布加锁
            c = holder.lockCanvas();
```

```
                    if (c != null)
                    {
                        // 绘制背景色
                        c.drawColor(0xffffffff);
                        // 在触碰点绘制心形
                        drawTouchPoint(c);
                        // 设置画笔的透明度
                        mPaint.setAlpha(76);
                        c.translate(originX, originY);
                        // 采用循环绘制 count 个矩形
                        for (int i = 0; i < count; i++)  //①
                        {
                            c.translate(80, 0);
                            c.scale(0.95f, 0.95f);
                            c.rotate(20f);
                            c.drawRect(0, 0, 150, 75, mPaint);
                        }
                    }
                }
                finally
                {
                    if (c != null) holder.unlockCanvasAndPost(c);
                }
                mHandler.removeCallbacks(drawTarget);
                // 调度下一次重绘
                if (mVisible)
                {
                    count ++;
                    if(count >= 50)
                    {
                        Random rand = new Random();
                        count = 1;
                        originX += (rand.nextInt(60) - 30);
                        originY += (rand.nextInt(60) - 30);
                        try
                        {
                            Thread.sleep(500);
                        }
                        catch (InterruptedException e)
                        {
                            e.printStackTrace();
                        }
                    }
                    // 指定 0.1 秒后重新执行 mDrawCube 一次
                    mHandler.postDelayed(drawTarget, 100);  //②
                }
            }
            // 在屏幕触碰点绘制圆圈
            private void drawTouchPoint(Canvas c)
            {
                if (mTouchX >= 0 && mTouchY >= 0)
                {
                    // 设置画笔的透明度
                    mPaint.setAlpha(255);
                    c.drawBitmap(heart , mTouchX, mTouchY, mPaint);
                }
            }
        }
    }
```

上面的程序中粗体字代码就是实现动态壁纸 Service 的关键代码，这两段粗体字代码重

写了 WallpaperService.Engine 的 onVisibilityChanged()、onOffsetsChanged()方法，并指定当桌面显示时调用 drawFrame()方法进行绘制，drawFrame()方法绘制完成后通过 Handler 对象指定 0.1 秒后重绘，如上面的程序中②号粗体字代码所示。

上面的程序为了实现“蜿蜒前行”的效果，使用循环控制绘制了 count 个矩形，每绘制一次，count 的值将会加 1，而且程序控制对 Canvas 进行位移、旋转两种坐标变换，通过这种方式即可实现“蜿蜒前行”的动画效果。

定义了该 Service 类之后，接下来还需要在 AndroidManifest.xml 文件中配置该 Service。配置动态壁纸 Service 与配置普通 Service 存在小小的区别。它需要指定如下两项。

- 指定运行动态壁纸，需要 android.permission.BIND_WALLPAPER 权限。
- 为动态壁纸指定 meta-data 配置。

在 AndroidManifest.xml 文件中配置动态壁纸，也就是需要增加如下配置片段。

程序清单：codes\14\14.2\LiveWallPaper\AndroidManifest.xml

```
<!-- 配置动态壁纸 Service -->
<service android:label="@string/app_name"
    android:name=".LiveWallpaper"
    android:permission="android.permission.BIND_WALLPAPER">
    <!-- 为动态壁纸配置 intent-filter -->
    <intent-filter>
        <action android:name="android.service.wallpaper.WallpaperService" />
    </intent-filter>
    <!-- 为动态壁纸配置 meta-data -->
    <meta-data android:name="android.service.wallpaper"
        android:resource="@xml/livewallpaper" />
</service>
```

上面的配置文件中粗体字配置部分就是配置动态壁纸的关键代码。上面的配置文件中指定了动态壁纸的 meta-data 放在@xml/livewallpaper 中定义，因此程序还需要在 res\xml\目录下增加一个 livewallpaper.xml 文件，该文件代码如下。

程序清单：codes\14\14.2\LiveWallPaper\res\xml\livewallpaper.xml

```
<?xml version="1.0" encoding="utf-8"?>
<wallpaper xmlns:android="http://schemas.android.com/apk/res/android"/>
```

把上面的程序部署到模拟器上，该程序不能直接运行，需要按前面介绍的步骤来设置动态壁纸。在图 14.6 所示列表中单击“Live Wallpapers”项，系统将会显示如图 14.7 所示的列表项。



图 14.7 选择动态壁纸

单击图 14.7 所示的“蜿蜒前行”项，系统进入预览该动态壁纸的界面，如图 14.8 所示。

在图 14.8 所示的界面中可以看到“蜿蜒前行”的动画不行执行，单击图 14.8 所示界面中的“Set wallpaper”按钮，即可应用我们刚刚所开发的动态壁纸程序。再次切换到 Android 系统桌面，将可看到桌面上显示如图 14.9 所示的效果。

虽然上面的程序介绍的动态壁纸只是在桌面上绘制蜿蜒前行的矩形，但这种动态壁纸可以提供开发者在桌面上自由绘图的能力，因此到底要在系统桌面上绘制什么，完全取决于用户自己的选择。



图 14.8　预览动态壁纸



图 14.9　动态壁纸效果

14.3　通过程序添加快捷方式

对于一个希望拥有更多用户的应用来说，用户桌面可以说是所有软件的必争之地，如果用户在手机桌面上建立了该软件的快捷方式，用户将会更频繁地使用该软件。因此，所有 Android 程序都应该允许用户把软件的快捷方式添加到桌面上。

在程序中把一个软件的快捷方式添加到桌面上，只需要如下三步即可：

① 创建一个添加快捷方式的 Intent，该 Intent 的 Action 属性值应该为 com.android.launcher.action.INSTALL_SHORTCUT。

② 通过为该 Intent 添加 Extra 属性来设置快捷方式的标题、图标及快捷方式对应启动的程序。

③ 调用 sendBroadcast()方法发送广播即可添加快捷方式。



提示：

就像 Windows 时代，用户桌面是“兵家”必争之地，所有应用总是试图在用户桌面上创建自己的快捷方式，这样就可以让用户时时刻刻看到自己的程序，让用户经常使用自己的程序，从而依赖自己的程序。但需要注意的是，也有些用户对这种行为比较反感，他们喜欢自己的桌面简简单单，因此他们会把这些程序添加的图标删除，甚至会直接把程序卸载，所以希望读者注意，不要太激进，引起用户反感会导致程序被删除。

实例：让程序占领桌面

下面的程序是对第 7 章一个动画程序的修改，该程序提供了一个按钮，用户单击该按钮即可在桌面上建立该程序的快捷方式。程序代码如下。

程序清单：codes\14\14.3\Add Shortcut\src\org\crazyit\desktop\Add Shortcut.java

```
public class AddShortcut extends Activity
{
    ImageView flower;
    // 定义两份动画资源
    Animation anim, reverse;
    final Handler handler = new Handler()
    {
```

```
        @Override
        public void handleMessage(Message msg)
        {
            if (msg.what == 0x123)
            {
                flower.startAnimation(reverse);
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        flower = (ImageView) findViewById(R.id.flower);
        // 加载第一份动画资源
        anim = AnimationUtils.loadAnimation(this, R.anim.anim);
        // 设置动画结束后保留结束状态
        anim.setFillAfter(true);
        // 加载第二份动画资源
        reverse = AnimationUtils.loadAnimation(this, R.anim.reverse);
        // 设置动画结束后保留结束状态
        reverse.setFillAfter(true);
        Button bn = (Button) findViewById(R.id.bn);
        // 为按钮的单击事件添加监听器
        bn.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 创建添加快捷方式的 Intent
                Intent addIntent = new Intent(
                    "com.android.launcher.action.INSTALL_SHORTCUT"); //①
                String title = getResources().getString(R.string.title);
                // 加载快捷方式的图标
                Parcelable icon = Intent.ShortcutIconResource.fromContext(
                    AddShortcut.this, R.drawable.ic_launcher);
                // 创建点击快捷方式后操作 Intent，该处当点击创建的快捷方式后，再次启
                // 动该程序
                Intent myIntent = new Intent(AddShortcut.this,
                    AddShortcut.class);
                // 设置快捷方式的标题
                addIntent.putExtra(Intent.EXTRA_SHORTCUT_NAME, title); //②
                // 设置快捷方式的图标
                addIntent.putExtra(Intent.EXTRA_SHORTCUT_ICON_RESOURCE
                    , icon); //②
                // 设置快捷方式对应的 Intent
                addIntent.putExtra(Intent.EXTRA_SHORTCUT_INTENT
                    , myIntent); //②
                // 发送广播添加快捷方式
                sendBroadcast(addIntent); //③
            }
        });
    }
    @Override
    public void onResume()
    {
        super.onResume();
        // 开始执行动画
        flower.startAnimation(anim);
        // 设置 3.5 秒后启动第二个动画
```

```
            new Timer().schedule(new TimerTask()
            {
                @Override
                public void run()
                {
                    handler.sendEmptyMessage(0x123);
                }
            }, 3500);
        }
    }
```

上面的程序中粗体字代码就是为程序添加快捷方式的关键代码，其中程序中①、②、③号代码就是前面所介绍的第 1 步、第 2 步、第 3 步。

在程序中添加快捷方式需要相应的权限，因此别忘了在 AndroidManifest.xml 文件中添加如下配置片段：

```
<!-- 指定添加安装快捷方式的权限 -->
<uses-permission android:name=
    "com.android.launcher.permission.INSTALL_SHORTCUT"/>
```



图 14.10　快捷方式

运行该程序，并单击该程序中的“添加快捷键”按钮，即可在桌面上添加一个快捷方式。返回桌面，即可在桌面上看到如图 14.10 所示的快捷方式。

上面的实例把添加快捷方式的代码放在事件处理方法中完成，这表明只有当用户单击该按钮时才会添加快捷方式——这种方式比较温和；如果开发者希望强行在用户桌面上添加快捷方式，可以将上面添加快捷方式的代码放在 onCreate()方法中——这种方式容易引起用户反感。

14.4　管理桌面控件

所谓桌面控件，就是指能直接显示在 Android 系统桌面的小程序，比如图 14.1 所看到的直接显示在桌面上的模拟时钟。一般来说，开发者可以把一些用户使用十分频繁的程序，比如时钟、指南针、日历等程序做成桌面控件，这样用户可以直接在桌面上看到程序的运行界面。

➤➤14.4.1　开发桌面控件

桌面控件是通过 Broadcast 的形式来进行控制的，因此每个桌面控件都对应于一个 BroadcastReceiver。为了简化桌面控件的开发，Android 系统提供了一个 AppWidgetProvider 类，它就是 BroadcastReceiver 的子类，也就是说开发者开发桌面控件只要继承 AppWidgetProvider 类即可。

为了开发桌面控件，开发者只要开发一个继承 AppWidgetProvider 的子类，并重写 AppWidgetProvider 不同状态的生命周期方法即可。AppWidgetProvider 里提供如下 4 个不同的生命周期方法。

- onUpdate()：负责更新桌面控件的方法；实现桌面控件通常会考虑重写该方法。
- onDeleted()：当一个或多个桌面控件被删除时回调该方法。
- onEnabled()：当接收到 ACTION_APPWIDGET_ENABLED Broadcast 时回调该方法。

- onDisabled()：当接收到 ACTION_APPWIDGET_DISABLED Broadcast 时回调该方法。

一般来说，开发桌面控件只需要定义一个 AppWidgetProvider 的子类，并重写它的 onUpdate()方法即可，重写该方法按如下步骤进行。

1. 创建一个 RemoteViews 对象，创建该对象时可以指定加载指定的界面布局文件。
2. 如果需要改变上一步所加载的界面布局文件的的内容，可通过 RemoteViews 对象进行修改。



提示：

一般来说，RemoteViews 所加载的界面中主要包含 ImageView 和 TextView 两种组件，RemoteViews 提供了修改这两种组件的内容的方法。

3. 创建一个 ComponentName 对象。
4. 调用 AppWidgetManager 更新桌面控件。



提示：

将上面 4 个步骤归纳起来，其核心代码就是使用 AppWidgetManager 通过 RemteViews 来更新 AppWidgetProvider 的子类实例（需将它包装成 ComponentName 对象）。

下面的示例程序中的桌面控件十分简单，该控件中只是包含了一张简单的图片，此处不再给出界面布局文件。添加桌面控件的代码如下。

程序清单：codes\14\14.4\DesktopApp\src\org\crazyit\desktop\DesktopApp.java

```
public class DesktopApp extends AppWidgetProvider
{
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
        int[] appWidgetIds)
    {
        // 加载指定界面布局文件，创建 RemoteViews 对象
        RemoteViews remoteViews = new RemoteViews(
            context.getPackageName(),
            R.layout.main); //①
        // 为 show ImageView 设置图片
        remoteViews.setImageViewResource(R.id.show
            , R.drawable.logo); //②
        // 将 AppWidgetProvider 子类实例包装成 ComponentName 对象
        ComponentName componentName = new ComponentName(
            context, DesktopApp.class); //③
        // 调用 AppWidgetManager 将 remoteViews 添加到 ComponentName 中
        appWidgetManager.updateAppWidget(componentName
            , remoteViews); //④
    }
}
```

上面的程序中重写了 AppWidgetProvider 的 onUpdate()方法，程序中①、②、③、④号代码正好对应于上面介绍的 4 个步骤。

由于 AppWidgetProvider 继承了 BroadcastReceiver，因此 AppWidgetProvider 的本质还是一个 BroadcastReceiver，为此需要在 AndroidManifest.xml 文件中使用<receiver.../>元素来配置它，配置该元素时需要为它指定相应的<intent-filter.../>和<meta-data.../>。

为了在系统中添加该桌面控件，还需要在AndroidManifest.xml文件中添加如下配置片段：

```
<receiver android:name="DesktopApp"
      android:label="@string/app_name">
      <!-- 将该BroadcastReceiver当成桌面控件 -->
      <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
      </intent-filter>
      <!-- 指定桌面控件的meta-data -->
      <meta-data android:name="android.appwidget.provider"
            android:resource="@xml/appwidget_provider"/>
</receiver>
```

上面的配置文件的粗体字代码指定该桌面控件使用@xml/my_appwidget作为meta-data，因此还需要在应用的res\xml目录下添加appwidget_provider.xml文件。该文件的内容如下。

程序清单：codes\14\14.4\DesktopApp\res\xml\appwidget_provider.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 指定该桌面组件的基本配置信息：
      minWidth：桌面控件的最小宽度。
      minWidth：桌面控件的最小高度。
      updatePeriodMillis：更新频率
      initialLayout：初始时显示的布局 -->
<appwidget-provider
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:minWidth="150dip"
      android:minHeight="70dip"
      android:updatePeriodMillis="1000"
      android:initialLayout="@layout/main"/>
```

上面的XML配置文件中使用了<appwidget-provider.../>元素来描述桌面控件的基本信息，上面的注释已经详细介绍了各属性的作用。

把该应用安装到Android系统上，再次进入图14.4所示的Widgets列表，将会看到如图14.11所示的界面。

正如图14.11所示，刚才开发的桌面控件已经显示出来了，长按该列表项就会把该小控件添加到桌面上。再次返回桌面将会看到如图14.12所示的桌面控件。



图14.11　添加桌面控件



图14.12　桌面控件

从图14.12可以看出，通过使用桌面控件，开发者可以把应用程序的运行界面直接放到桌面上，至于如何控制程序界面上显示的内容，完全取决于项目的业务需求。下面以开发一个液晶时钟为例，来介绍桌面控件的用法。

实例：液晶时钟

为了实现一个液晶时钟的桌面组件，开发者需要在程序界面上定义8个ImageView，其

中 6 个 ImageView 用于显示小时、分钟、秒钟的数字，另外两个 ImageView 用于显示小时、分钟、秒钟之间的冒号。

为了让桌面组件实时地显示当前时间，程序需要每隔 1 秒更新一次程序界面上的 6 个 ImageView，让它们显示当前小时、分钟、秒钟的数字即可。

液晶时钟的代码如下。

程序清单：codes\14\14.4\LedClock\src\org\crazyit\desktop\LedClock.java

```
public class LedClock extends AppWidgetProvider
{
    private Timer timer = new Timer();
    private AppWidgetManager appWidgetManager;
    private Context context;
    // 将 0～9 的液晶数字图片定义成数组
    private int[] digits = new int[]
    {
        R.drawable.su01,
        R.drawable.su02,
        R.drawable.su03,
        R.drawable.su04,
        R.drawable.su05,
        R.drawable.su06,
        R.drawable.su07,
        R.drawable.su08,
        R.drawable.su09,
        R.drawable.su10,
    };
    // 将显示小时、分钟、秒钟的 ImageView 定义成数组
    private int[] digitViews = new int[]
    {
        R.id.img01,
        R.id.img02,
        R.id.img04,
        R.id.img05,
        R.id.img07,
        R.id.img08
    };
    @Override
    public void onUpdate(Context context, AppWidgetManager appWidgetManager,
            int[] appWidgetIds)
    {
        System.out.println("--onUpdate--");
        this.appWidgetManager = appWidgetManager;
        this.context = context;
        // 定义计时器
        timer = new Timer();
        // 启动周期性调度
        timer.schedule(new TimerTask()
        {
            public void run()
            {
                // 发送空消息，通知界面更新
                handler.sendEmptyMessage(0x123);
            }
        }, 0, 1000);
    }
    private Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
```

```
                if (msg.what == 0x123)
                {
                    RemoteViews views = new RemoteViews(context
                            .getPackageName(), R.layout.main);
                    // 定义 SimpleDateFormat 对象
                    SimpleDateFormat df = new SimpleDateFormat(
                            "HHmmss");
                    // 将当前时间格式化成 HHmmss 的形式
                    String timeStr = df.format(new Date());
                    for(int i = 0 ; i < timeStr.length() ;i++)
                    {
                        // 将第 i 个数字字符转换为对应的数字
                        int num = timeStr.charAt(i) - 48;
                        // 将第 i 个图片设为对应的液晶数字图片
                        views.setImageViewResource(digitViews[i], digits[num]);
                    }
                    // 将 AppWidgetProvider 子类实例包装成 ComponentName 对象
                    ComponentName componentName = new ComponentName(context,
                            LedClock.class);
                    // 调用 AppWidgetManager 将 remoteViews 添加到 ComponentName 中
                    appWidgetManager.updateAppWidget(componentName, views);
                }
                super.handleMessage(msg);
            }
        };
    }
```

上面的程序中粗体字代码将会根据程序的时间字符串动态更新 6 个 ImageView 所显示的液晶数字图片，这样即可通过液晶数字来显示当前时间了。

在 AndroidManifest.xml 文件中添加如下代码片段来定义该桌面控件：

```
<receiver android:name=".LedClock"
        android:label="@string/app_name">
        <!-- 将该 BroadcastReceiver 当成桌面控件 -->
        <intent-filter>
            <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>
        <!-- 指定桌面控件的 meta-data -->
        <meta-data android:name="android.appwidget.provider"
            android:resource="@xml/appwidget_provider"/>
</receiver>
```

上面的配置文件中粗体字代码指定了该液晶时钟的 meta-data 为 @xml/appwidget_ provider，还需要在 res\xml 路径下增加一个 appwidget_provider.xml 文件，该文件的内容与前一个桌面控件的 meta-data 文件大致相同，此处不再给出。



图 14.13　液晶时钟控件

通过 Android 系统把该液晶时钟小控件添加到桌面上，将可以看到 Android 桌面显示如图 14.13 所示的液晶时钟。

正如图 14.13 所示液晶时钟所示，这种桌面控件显示了程序的运行状态，它的运行界面可以是动态改变的。因此还可以通过桌面控件把股票走势图之类添加到桌面上，让用户可以一眼看到。总之这种桌面控件给了开发者更多的想象空间。

14.4.2　Android 4.0 新增的显示数据集的桌面控件

Android 3.0 为 RemoteViews 新增了如下方法。

setRemoteAdapter (int viewId, Intent intent)：该方法可以使用 Intent 更新 RemoteViews 中 viewId 对应的组件。



提示： 从 Android 4.0 开始，Android 不再支持本书第 1 版所介绍的 LiveFolder，改为使用显示数据集的桌面控件来代替 LiveFolder，因此本书也删除了第 1 版中关于 LiveFolder 的内容。

上面方法的 Intent 参数应该封装一个 RemoteViewsService 参数，RemoteViewsService 虽然继承了 Service 组件，但它的主要作用是为 RemoteViews 中 viewId 对应的组件提供列表项。

由于 Intent 参数负责提供列表项，因此 viewId 参数对应的组件可以是 ListView、GridView、StackView 和 AdapterViewFlipper 等，这些组件都是 AdapterView 的子类，由此可见 RemoteViewsService 负责提供的对象，应该是一个类似于 Adapter 的对象。

RemoteViewsService 通常用于被继承，继承该基类时需要重写它的 onGetViewFactory() 方法，该方法就需要返回一个类似于 Adapterr 对象——但不是 Adapter，而是 RemoteViewsFactory 对象，RemoteViewsFactory 的功能完全类似于 Adapter。

下面是本示例提供的 RemoteViewsService 类。

程序清单：codes\14\14.4\AppWidgetCollection\src\org\crazyit\desktop\StackWidgetService. java

```
public class StackWidgetService extends RemoteViewsService
{
    // 重写该方法，该方法返回一个 RemoteViewsFactory 对象
    // RemoteViewsFactory 对象的作用类似于 Adapter
    // 它负责为 RemoteView 中的指定组件提供多个列表项
    @Override
    public RemoteViewsFactory onGetViewFactory(Intent intent)
    {
        return new StackRemoteViewsFactory(this.getApplicationContext(),
            intent); //①
    }
    class StackRemoteViewsFactory implements
    RemoteViewsService.RemoteViewsFactory
    {
        // 定义一个数组来保存该组件生成的多个列表项
        private int[] items = null;
        private Context mContext;
        public StackRemoteViewsFactory(Context context, Intent intent)
        {
            mContext = context;
        }
        @Override
        public void onCreate()
        {
            // 初始化 items 数组
            items = new int[] { R.drawable.bomb5, R.drawable.bomb6,
                R.drawable.bomb7, R.drawable.bomb8, R.drawable.bomb9,
                R.drawable.bomb10, R.drawable.bomb11, R.drawable.bomb12,
                R.drawable.bomb13, R.drawable.bomb14, R.drawable.bomb15,
                R.drawable.bomb16
            };
        }
        @Override
        public void onDestroy()
        {
```

```
            items = null;
        }
        // 该方法的返回值控制该对象包含多少个列表项
        @Override
        public int getCount()
        {
            return items.length;
        }
        // 该方法的返回值控制各位置所显示的 RemoteViews
        @Override
        public RemoteViews getViewAt(int position)
        {
            // 创建 RemoteViews 对象，加载/res/layout 目录下的 widget_item.xml 文件
            RemoteViews rv = new RemoteViews(mContext.getPackageName(),
                R.layout.widget_item);
            // 更新 widget_item.xml 布局文件中的 widget_item 组件
            rv.setImageViewResource(R.id.widget_item,
                items[position]);
            // 创建 Intent、用于传递数据
            Intent fillInIntent = new Intent();
            fillInIntent.putExtra(StackWidgetProvider.EXTRA_ITEM, position);
            // 设置当单击该 RemoteViews 时传递 fillInIntent 包含的数据
            rv.setOnClickFillInIntent(R.id.widget_item, fillInIntent);
            // 此处让线程暂停 0.5 秒来模拟加载该组件
            try
            {
                System.out.println("加载【" + position + "】位置的组件");
                Thread.sleep(500);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
            return rv;
        }
        @Override
        public RemoteViews getLoadingView()
        {
            return null;
        }
        @Override
        public int getViewTypeCount()
        {
            return 1;
        }
        @Override
        public long getItemId(int position)
        {
            return position;
        }
        @Override
        public boolean hasStableIds()
        {
            return true;
        }
        @Override
        public void onDataSetChanged()
        {
        }
    }
}
```

上面的程序中①号粗体字代码返回了一个 StackRemoteViewsFactory 对象，该对象的作用类似于 Adapter 对象，它的作用就是负责返回多个列表项。与普通 Adapter 不同的是，Adapter 返回的多个列表项只要是 View 组件即可；但 StackRemoteViewsFactory 对象返回的列表项必须是 RemoteViews 组件，因此上面的程序中粗体字代码重写 getViewAt(int position)方法时需要创建并返回一个 RemoteViews 对象。

上面的程序加载的布局文件为位于/res/layout 目录下的 widget_item.xml，该文件只包含一个简单的 ImageView 组件。该布局文件代码如下：

程序清单：codes\14\14.4\AppWidgetCollection\res\layout\widget_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ImageView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget_item"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:gravity="center"/>
```

提供了 StackWidgetService 之后，接下来开发 AppWidgetProvider 的子类与开发普通 AppWidgetProvider 的子类的步骤基本相同，只是此时不再调用 RemoteViews 的 setImageViewResource()或 setTextViewText()，而是调用 setRemoteAdapter(int viewId, Intent intent)方法。

下面是本示例中 AppWidgetProvider 子类的代码。

程序清单：codes\14\14.4\AppWidgetCollection\src\org\crazyit\desktop\StackWidgetProvider. java

```
public class StackWidgetProvider extends AppWidgetProvider
{
    public static final String TOAST_ACTION
        = "org.crazyit.desktop.TOAST_ACTION";
    public static final String EXTRA_ITEM
        = "org.crazyit.desktop.EXTRA_ITEM";
    @Override
    public void onUpdate(Context context,
        AppWidgetManager appWidgetManager, int[] appWidgetIds)
    {
        // 创建 RemoteViews 对象，加载/res/layout 目录下的 widget_layout.xml 文件
        RemoteViews rv = new RemoteViews(context.getPackageName(),
            R.layout.widget_layout);
        Intent intent = new Intent(context, StackWidgetService.class);
        // 使用 intent 更新 rv 中的 stack_view 组件（StackView）
        rv.setRemoteAdapter(R.id.stack_view, intent); //①
        // 设置当 StackWidgetService 提供的列表项为空时，直接显示 empty_view 组件
        rv.setEmptyView(R.id.stack_view, R.id.empty_view);
        // 创建启动 StackWidgetProvider 组件（作为 BroadcastReceiver）的 Intent
        Intent toastIntent = new Intent(context,
            StackWidgetProvider.class);
        // 为该 Intent 设置 Action 属性
        toastIntent.setAction(StackWidgetProvider.TOAST_ACTION);
        // 将 Intent 包装成 PendingIntent
        PendingIntent toastPendingIntent = PendingIntent
            .getBroadcast(context, 0, toastIntent,
                PendingIntent.FLAG_UPDATE_CURRENT);
        // 将 PendingIntent 与 stack_view 进行关联
        rv.setPendingIntentTemplate(R.id.stack_view,
            toastPendingIntent);
        // 使用 AppWidgetManager 通过 RemteViews 更新 AppWidgetProvider
```

```
            appWidgetManager.updateAppWidget(
                  new ComponentName(context, StackWidgetProvider.class), rv); //②
            super.onUpdate(context, appWidgetManager, appWidgetIds);
      }
      @Override
      public void onDeleted(Context context, int[] appWidgetIds)
      {
            super.onDeleted(context, appWidgetIds);
      }

      @Override
      public void onDisabled(Context context)
      {
            super.onDisabled(context);
      }
      @Override
      public void onEnabled(Context context)
      {
            super.onEnabled(context);
      }
      // 重写该方法，将该组件当成 BroadcastReceiver 使用
      @Override
      public void onReceive(Context context, Intent intent)
      {
            if (intent.getAction().equals(TOAST_ACTION))
            {
                  // 获取 Intent 中的数据
                  int viewIndex = intent.getIntExtra(EXTRA_ITEM, 0);
                  // 显示 Toast 提示
                  Toast.makeText(context, "点击第【" + viewIndex + "】个列表项",
                        Toast.LENGTH_SHORT).show();
            }
            super.onReceive(context, intent);
      }
}
```

上面①号粗体字代码调用 RemoteViews 的 setRemoteAdapter()方法进行设置，该方法负责为 RemoteViews 中 StackView 设置 RemoteViewsFactory 对象——该 RemoteViewsFactory 对象负责提供多个列表项。接下来程序中的②号粗体字代码调用 AppWidgetManager 的 updateAppWidget()进行更新，这行代码与开发普通的 AppWidgetProvider 基本相同。

由于这种带数据集的桌面控件同时需要 AppWidgetProvider 和 RemoteViewsService，因此在 AndroidManifest.xml 文件中需要同时配置<receiver.../>元素和<service.../>元素。本示例的配置片段如下。

程序清单：codes\14\14.4\AppWidgetCollection\AndroidManifest.xml

```
<application
      android:allowBackup="true"
      android:label="@string/app_name">
      <!-- 配置 AppWidgetProvider，即配置桌面控件 -->
      <receiver android:name=".StackWidgetProvider">
         <!-- 通过该 intent-filter 指定该 Receiver 作为桌面控件 -->
          <intent-filter>
                <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
          </intent-filter>
          <!-- 为桌面控件指定 meta-data -->
          <meta-data
                android:name="android.appwidget.provider"
                android:resource="@xml/stackwidgetinfo" />
```

```
        </receiver>
        <!-- 配置 RemoteViewsService
        必须指定权限为 android.permission.BIND_REMOTEVIEWS
         -->
        <service
              android:name=".StackWidgetService"
              android:permission="android.permission.BIND_REMOTEVIEWS"
              android:exported="false" />
    </application>
```

该实例同样还需要在/res/xml 目录下添加一个 stackwidgetinfo.xml 元数据配置文件，该文件与前面开发普通 AppWidgetProvider 基本相似，此处不再给出。

将该应用部署到模拟器上，并将该桌面控件添加到桌面上，将可以看到如图 14.14 所示的桌面控件。



图 14.14 带数据集的桌面控件

14.5 本章小结

Android 系统的桌面是用户每天接触最多的界面，如果用户把我们开发的应用添加到系统桌面上，可以大大增加用户对该软件的依赖。本章主要介绍了如何管理 Android 手机桌面，包括开发动态壁纸、管理手机桌面上的快捷方式、管理桌面小控件、带数据集的桌面小控件等，这些内容是读者需要重点掌握的。

CHAPTER

15

第15章 传感器应用开发

本章要点

- 手机传感器的概念和作用
- 定义监听器监听传感器的数据
- 加速度传感器
- 方向传感器
- 磁场传感器
- 温度传感器
- 光传感器
- 压力传感器
- 使用方向传感器开发指南针
- 使用方向传感器开发水平仪

Android 系统提供了对传感器的支持，如果手机设备的硬件提供了这些传感器，Android 应用可以通过传感器来获取设备的外界条件，包括手机设备的运行状态、当前摆放方向、外界的磁场、温度和压力等。Android 系统提供了驱动程序去管理这些传感器硬件，当传感器硬件感知到外部环境发生改变时，Android 系统负责管理这些传感器数据。

对于 Android 应用开发者来说，开发传感器应用十分简单，开发者只要为指定监听器注册一个监听器即可，当外部环境发生改变时，Android 系统会通过传感器获取外部环境的数据，并将数据传给监听器的监听方法。

通过在 Android 应用中添加传感器，可以充分激发开发者、用户的想象力，可以开发出各种新奇的程序，比如电子罗盘、水平仪等；除此之外，还可以利用传感器开发各种游戏，必须通过传感器来感知用户动作，从而在游戏中提供对应的响应。

15.1 利用 Android 的传感器

在 Android 平台上开发传感器应用十分简单，下面通过一个简单的加速度传感器来介绍传感器应用的开发。

15.1.1 开发传感器应用

在 Android 系统中开发传感器应用十分简单，因为 Android 系统为传感器支持强大的管理服务。开发传感器应用的步骤如下：

① 调用 Context 的 getSystemService(Context.SENSOR_SERVICE)方法获取 SensorManager 对象，SensorManager 对象代表系统的传感器管理服务。

② 调用 SensorManager 的 getDefaultSensor(int type)方法来获取指定类型的传感器。

③ 通常选择在 Activity 的 onResume()方法中调用 SensorManager 的 registerListener()为指定传感器注册监听即可。程序通过实现监听器即可获取传感器传回来的数据。

SensorManager 提供的注册传感器的方法为：registerListener(SensorEventListener listener, Sensor sensor, int rate)，该方法中三个参数的说明如下。

- listener：监听传感器事件的监听器。该监听器需要实现 SensorEventListener 接口。
- sensor：传感器对象。
- rate：指定获取传感器数据的频率。

该方法中 rate 可以获取传感器数据的频率，它支持如下几个频率值。

- SensorManager.SENSOR_DELAY_FASTEST：最快。延迟最小，只有特别依赖于传感器数据的应用推荐采用这种频率，该种模式可能造成手机电量大量消耗，由于传递的为原始数据，算法不处理好将会影响应用的性能。
- SensorManager.SENSOR_DELAY_GAME：适合游戏的频率。在一般实时性要求的应用上适合使用这种频率。
- SensorManager.SENSOR_DELAY_NORMAL：正常频率。一般实时性要求不是特别高的应用上适合这种频率。
- SensorManager.SENSOR_DELAY_UI：适合普通用户界面的频率。这种模式比较省电、而且系统开销也很小，但延迟较大，因此只适合在普通小程序中使用。

下面将会按上面的步骤来开发一个加速度传感器应用。该程序的界面很简单，提供一个

文本框来显示加速度值即可，此处不再给出界面布局代码。该应用的 Activity 代码如下。

程序清单：codes\15\15.1\AccelerometerTest\src\org\crazyit\sensor\AccelerometerTest.java

```
public class AccelerometerTest extends Activity
      implements SensorEventListener
{
      // 定义系统的 Sensor 管理器
      SensorManager sensorManager;
      EditText etTxt1;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取程序界面上的文本框组件
            etTxt1 = (EditText) findViewById(R.id.txt1);
            // 获取系统的传感器管理服务
            sensorManager = (SensorManager) getSystemService(
                  Context.SENSOR_SERVICE);  //①
      }
      @Override
      protected void onResume()
      {
            super.onResume();
            // 为系统的加速度传感器注册监听器
            sensorManager.registerListener(this,
                  sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                  SensorManager.SENSOR_DELAY_GAME);  //②
      }
      @Override
      protected void onStop()
      {
            // 取消注册
            sensorManager.unregisterListener(this);
            super.onStop();
      }
      // 以下是实现 SensorEventListener 接口必须实现的方法
      // 当传感器的值发生改变时回调该方法
      @Override
      public void onSensorChanged(SensorEvent event)
      {
            float[] values = event.values;
            StringBuilder sb = new StringBuilder();
            sb.append("X 方向上的加速度：");
            sb.append(values[0]);
            sb.append("\nY 方向上的加速度：");
            sb.append(values[1]);
            sb.append("\nZ 方向上的加速度：");
            sb.append(values[2]);
            etTxt1.setText(sb.toString());
      }
      // 当传感器精度改变时回调该方法
      @Override
      public void onAccuracyChanged(Sensor sensor, int accuracy)
      {
      }
}
```

上面的程序中①号粗体字代码用于获取系统的传感器管理服务，②号粗体字代码则用于获取加速度传感器，并为该传感器注册监听器。本程序直接使用了 Activity 充当传感器监听

器，因此该 Activity 实现了 SensorEventListener 接口，并实现了该接口中的两个方法。

- onSensorChanged()：当传感器的值发生改变时触发该方法。
- onAccuracyChanged()：当传感器的精度发生改变时触发该方法。

上面的程序实现 onSensorChanged()方法时通过 SensorEvent 对象的 values()方法来获取传感器的值，不同传感器所返回的值的个数是不等的。对于加速度传感器来说，它将返回三个值，分别代表手机设备在 *X*、*Y*、*Z* 三个方向上的加速度。

需要指出的是，传感器的坐标系统与屏幕坐标系统不同，传感器坐标系统的 *X* 轴沿屏幕向左；*Y* 轴则沿屏幕向上，*Z* 轴则垂直于屏幕向里。图 15.1 显示了传感器的坐标系统。

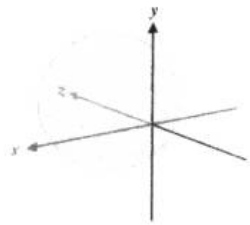


图 15.1 传感器的坐标系统

从图 15.1 的坐标系统大致可以看出，当拿着手机横向左右移动时，可能产生 *X* 轴上的加速度；拿着手机前后移动时，可能产生 *Y* 轴上的加速度；当拿着手机竖向上下移动时，可能产生 *Z* 轴上的加速度。

运行上面的程序，将会看到如图 15.2 所示的输出。

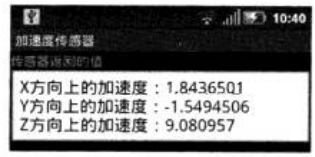


图 15.2 加速度模拟器

从图 15.2 可以看出，当用户拿着手机“晃动”时，即可看到程序能检测到传感器返回的加速度值。



备注：

Android 模拟器并默认没有提供传感器支持，因此本章的所有示例程序都应使用真机进行测试，才可看到正确的运行效果。本书第 1 版曾介绍使用 SensorSimulator 工具来模拟传感器，通过这种方式可以向 Android 模拟器增加传感器支持，但这种方法毕竟不够真实，而且目前 Android 手机已经大范围普及，出于篇幅考虑，本书第 2 版删除了使用 SensorSimulator 模拟传统器的内容，如果读者确实需要了解使用 SensorSimulator 模拟传感器的内容，建议参考本书第 1 版。

15.2 Android 的常用传感器

上一章介绍了如何监听 Android 设备的加速度，只要通过 SensorManager 为加速度传感器注册监听器即可。实际上 Android 系统对所有类型的传感器的处理完全一样，只是传感器的类型有所区别而已。

15.2.1 方向传感器 Orientation

方向传感器用于感应手机设备的摆放状态。方向传感器可以返回三个角度，这三个角度即可确定手机的摆放状态。

关于方向传感器返回的三个角度的说明如下。

- 第一个角度：表示手机顶部朝向与正北方的夹角。当手机绕着 *Z* 轴旋转时，该角度值发生改变。例如当该角度为 0 时，表明手机顶部朝向正北；该角度为 90 时，代表手机顶部朝向正东；该角度为 180 时，代表手机顶部朝向正南；该角度为 180 时，代表手机顶部朝向正西。

- 第二个角度：表示手机顶部或尾部翘起的角度。当手机绕着 X 轴倾斜时，该角度值发生变化。该角度的取值范围是-180～180。假设将手机屏幕朝上水平放在桌子上，如果桌子是完全水平的，该角度值应该是 0。假如从手机顶部开始抬起，直到将手机沿 X 轴旋转 180 度（屏幕向下水平放在桌面上），在这个旋转过程中，该角度值会从 0 变化到-180。也就是说，从手机顶部抬起时，该角度的值会逐渐减小，直到等于-180；如果从手机底部开始抬起，直到将手机沿 X 轴旋转 180 度（屏幕向下水平放在桌面上），该角度的值会从 0 变化到 180。也就是说，从手机底部抬起时，该角度的值会逐渐增大，直到等于 180。
- 第三个角度：表示手机左侧或右侧翘起的角度。当手机绕着 Y 轴倾斜时，该角度值发生变化。该角度的取值范围是-90～90。假设将手机屏幕朝上水平放在桌面上，如果桌面是完全水平的，该角度值应为 0。假如将手机左侧逐渐抬起，直到将手机沿 Y 轴旋转 90 度（手机与桌面垂直），在这个旋转过程中，该角度值会从 0 变化到-90。也就是说，从手机左侧抬起时，该角度的值会逐渐减小，直到等于-90；如果从手机右侧开始抬起，直到将手机沿 Y 轴旋转 90 度（手机与桌面垂直），该角度的值会从 0 变化到 90。也就是说，从手机右侧抬起时，该角度的值会逐渐增大，直到等于 90。

通过在应用程序中使用方向传感器，应用程序就可检测到手机设备的摆放状态：比如手机顶部的朝向，手机目前的倾斜角度等；借助于方向传感器，可以开发出指南针、水平仪等有趣的应用。

此处不单独介绍方向传感器的应用，后面通过一个示例来集中介绍所有传感器的用法示例。

15.2.2 磁场传感器 Magnetic Field

磁场传感器主要用于读取手机设备外部的磁场强度。即使周围没有任何直接的磁场，手机设备也始终会处于地球磁场中。随着手机设备摆放状态的改变，周围磁场在手机的 X、Y、Z 方向上的影响会发生改变。

磁场传感器会返回三个数据，三个数据分别代表周围磁场分解到 X、Y、Z 三个方向上的磁场分量，磁场数据的单位是微特斯拉（uT）。

此处不单独介绍磁场传感器的应用，后面通过一个示例来集中介绍所有传感器的用法示例。

15.2.3 温度传感器 Temperature

温度传感器用于获取手机设备所处环境的温度。

温度传感器会返回一个数据，该数据代表手机设备周围的温度，单位是摄氏度。

此处不单独介绍温度传感器的应用，后面通过一个示例来集中介绍所有传感器的用法示例。

15.2.4 光传感器 Light

光传感器用于获取手机设备所处环境的光的强度。

光传感器会返回一个数据，代表手机设备周围的光的强度，该数据的单位是勒克斯(lux)。

此处不单独介绍光传感器的应用，后面通过一个示例来集中介绍所有传感器的用法示例。

15.2.5 压力传感器 Pressure

压力传感器用于获取手机设备所处环境的压力的大小。

压力传感器会返回一个数据，代表手机设备周围的压力的大小。

此处不单独介绍压力传感器的应用，后面将通过一个示例来集中介绍所有传感器的用法示例。

正如前面介绍的，Android 系统对所有传感器的处理方式完全相同，接下来通过一个示例程序来介绍上面这些传感器的用法。该程序界面只是提供了几个文本框，分别用于显示不同的传感器数据。该程序代码如下。

程序清单：codes\15\15.2\SensorTest\src\org\crazyit\sensor\SensorTest.java

```
public class SensorTest extends Activity
    implements SensorEventListener
{
    // 定义 Sensor 管理器
    private SensorManager mSensorManager;
    EditText etOrientation;
    EditText etMagnetic;
    EditText etTemerature;
    EditText etLight;
    EditText etPressure;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面上的 EditText 组件
        etOrientation = (EditText) findViewById(R.id.etOrientation);
        etMagnetic = (EditText) findViewById(R.id.etMagnetic);
        etTemerature = (EditText) findViewById(R.id.etTemerature);
        etLight = (EditText) findViewById(R.id.etLight);
        etPressure = (EditText) findViewById(R.id.etPressure);
        // 获取传感器管理服务
        mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE); //①
    }
    @Override
    protected void onResume()
    {
        super.onResume();
        // 为系统的方向传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的磁场传感器注册监听器
        mSensorManager.registerListener(this,
            mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
            SensorManager.SENSOR_DELAY_GAME);
        // 为系统的温度传感器注册监听器
        mSensorManager.registerListener(this,
mSensorManager.getDefaultSensor(Sensor.TYPE_AMBIENT_TEMPERATURE),
```

```
                SensorManager.SENSOR_DELAY_GAME);
        // 为系统的光传感器注册监听器
        mSensorManager.registerListener(this,
                mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT),
                SensorManager.SENSOR_DELAY_GAME);
        // 为系统的压力传感器注册监听器
        mSensorManager.registerListener(this,
                mSensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE),
                SensorManager.SENSOR_DELAY_GAME);
    }
    @Override
    protected void onStop()
    {
        // 程序退出时取消注册传感器监听器
        mSensorManager.unregisterListener(this);
        super.onStop();
    }
    @Override
    protected void onPause()
    {
        // 程序暂停时取消注册传感器监听器
        mSensorManager.unregisterListener(this);
        super.onPause();
    }
    // 以下是实现 SensorEventListener 接口必须实现的方法
    @Override
    // 当传感器精度改变时回调该方法。
    public void onAccuracyChanged(Sensor sensor, int accuracy)
    {
    }
    @Override
    public void onSensorChanged(SensorEvent event)
    {
        float[] values = event.values;
        // 获取触发 event 的传感器类型
        int sensorType = event.sensor.getType();
        StringBuilder sb = null;
        // 判断是哪个传感器发生改变
        switch (sensorType)
        {
            // 方向传感器
            case Sensor.TYPE_ORIENTATION:
                sb = new StringBuilder();
                sb.append("绕 Z 轴转过的角度：");
                sb.append(values[0]);
                sb.append("\n 绕 X 轴转过的角度：");
                sb.append(values[1]);
                sb.append("\n 绕 Y 轴转过的角度：");
                sb.append(values[2]);
                etOrientation.setText(sb.toString());
                break;
            // 磁场传感器
            case Sensor.TYPE_MAGNETIC_FIELD:
                sb = new StringBuilder();
                sb.append("X 方向上的角度：");
                sb.append(values[0]);
                sb.append("\nY 方向上的角度：");
                sb.append(values[1]);
                sb.append("\nZ 方向上的角度：");
                sb.append(values[2]);
                etMagnetic.setText(sb.toString());
```

```
                    break;
                // 温度传感器
                case Sensor.TYPE_AMBIENT_TEMPERATURE:
                    sb = new StringBuilder();
                    sb.append("当前温度为：");
                    sb.append(values[0]);
                    etTemerature.setText(sb.toString());
                    break;
                // 光传感器
                case Sensor.TYPE_LIGHT:
                    sb = new StringBuilder();
                    sb.append("当前光的强度为：");
                    sb.append(values[0]);
                    etLight.setText(sb.toString());
                    break;
                // 压力传感器
                case Sensor.TYPE_PRESSURE:
                    sb = new StringBuilder();
                    sb.append("当前压力为：");
                    sb.append(values[0]);
                    etPressure.setText(sb.toString());
                    break;
            }
        }
    }
```

上面的程序一样遵守前面介绍的传感器编程步骤：①号代码在 Activity 的 onCreate()方法中获取 SensorManager 对象（如果在模拟器中调试，则获取模拟器对应的 SensorManager）；程序中大段的粗体字代码在 Activity 的 onResum()方法中为指定类型的传感器注册监听器，本程序为 5 种类型的传感器注册了监听器；实现 onSensorChanged(SensorEvent event)方法就实现传感器监听器，实现监听器方法时即可获取传感器所传回来的数据。

在真机中调试该程序，运行该程序，即可看到如图 15.3 所示的结果。

从图 15.3 可以看出，该程序并不能获取温度传感器和压力传感器的值，这是因为笔者的手机不支持温度传感器与压力传感器的缘故。



图 15.3 传感器应用

15.3 传感器应用案例

对传感器的支持是 Android 系统的特性之一，通过使用传感器可以轻易开发出各种有趣的应用，下面将会通过使用方向传感器来开发指南针和水平仪两个有趣的应用。

实例：指南针

前面介绍方向传感器时已经指出，水平传感器传回来的第一个参数值就是代表手机绕 Z 轴转过的角度，也就是手机顶部与正北的夹角，通过在程序中检测该夹角就可以开发出指南针。

开发指南针的思路很简单：程序先准备一张指南针图片，该图片上方向指针指向北方。接下来开发一个检测方向的传感器，程序检测到手机顶部绕 Z 轴转过多少度，让指南针图片反向转过多少度即可。由此可见指南针应用只要在界面中添加一张图片，并让图片总是反向

转过方向传感器返回的第一个角度值即可。下面是该程序的代码。

程序清单：codes\15\15.3\Compass\src\org\crazyit\sensor\Compass.java

```
public class Compass extends Activity
      implements SensorEventListener
{
      // 定义显示指南针的图片
      ImageView znzImage;
      // 记录指南针图片转过的角度
      float currentDegree = 0f;

       // 定义 Sensor 管理器
      SensorManager mSensorManager;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取界面中显示指南针的图片
            znzImage = (ImageView) findViewById(R.id.znzImage);
            // 获取传感器管理服务
            mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
      }
      @Override
      protected void onResume()
      {
            super.onResume();
            // 为系统的方向传感器注册监听器
            mSensorManager.registerListener(this,
                  mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
                  SensorManager.SENSOR_DELAY_GAME);
      }
      @Override
      protected void onPause()
      {
            // 取消注册
            mSensorManager.unregisterListener(this);
            super.onPause();
      }
      @Override
      protected void onStop()
      {
            // 取消注册
            mSensorManager.unregisterListener(this);
            super.onStop();
      }
      @Override
      public void onSensorChanged(SensorEvent event)
      {
            // 获取触发 event 的传感器类型
            int sensorType = event.sensor.getType();
            switch (sensorType)
            {
                  case Sensor.TYPE_ORIENTATION:
                        // 获取绕 Z 轴转过的角度
                        float degree = event.values[0];
                        // 创建旋转动画（反向转过 degree 度）
                        RotateAnimation ra = new RotateAnimation(currentDegree,
                              -degree, Animation.RELATIVE_TO_SELF, 0.5f,
                              Animation.RELATIVE_TO_SELF, 0.5f);
                        // 设置动画的持续时间
```

```
                    ra.setDuration(200);
                    // 运行动画
                    znzImage.startAnimation(ra);
                    currentDegree = -degree;
                    break;
                }
            }
            @Override
            public void onAccuracyChanged(Sensor sensor, int accuracy)
            {
            }
    }
```

指南针程序的关键代码就是程序中的粗体字代码，该程序检测到手机绕 Z 轴转过的角度，也就是手机 Y 轴与正北方向的夹角，然后让指南针图片反向转过相应的角度即可。



在真机中调试该程序，运行该程序，即可看到如图 15.4 所示的结果。

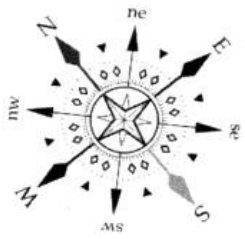


图 15.4 指南针

实例：水平仪

这里介绍的水平仪就是那种比较传统的水平仪，在一个的透明圆盘内中充满某种液体，液体中留有一个气泡，当一端翘起时，该气泡将会浮向翘起的一端。

前面介绍方向传感器时已经指出，方向传感器会返回三个角度值，其中第二个角度值代表底部翘起的角度（当顶部翘起时为负值）；第二个角度值代表右侧翘起的角度（当左侧翘起时为负值）；根据这两个角度值就可开发出水平仪了。

假设我们以大透明圆盘的中心为原点，当手机顶部翘起时，气泡应该向顶部移动，也就是气泡的位置的 Y 坐标（2D 绘图坐标系，屏幕左上角为原点）应减小；当手机底部翘起时，气泡应该向底部移动，也就是气泡的位置的 Y 坐标应增大——假设气泡开始位于大透明圆盘的中心，气泡的 Y 坐标的改变正好与方向传感器返回的第二个参数返回的角度的正负相符，因此根据方向传感器返回的第二个参数来计算气泡的 Y 坐标即可；与此类似，当手机左侧翘起时，气泡应该向左侧移动，也就是气泡的位置的 X 坐标（2D 绘图坐标系，屏幕左上角为原点）应减小；当手机右侧翘起时，气泡应该向右侧移动，也就是气泡的位置的 X 坐标应增大——假设气泡开始位于大透明圆盘的中心，气泡的 X 坐标的改变正好与方向传感器返回的第三个参数返回的角度的正负相符，因此根据方向传感器返回的第三个参数来计算气泡的 X 坐标即可。

通过上面介绍的方式来动态改变程序界面中气泡的位置——手机哪端翘起，水平仪中的气泡就浮向哪端，这就是水平仪的实现思想。

该程序用了一个自定义 View，该自定义 View 很简单，就是绘制透明圆盘和气泡——其中气泡的位置会动态改变。该自定义 View 的代码如下。

程序清单：codes\15\15.3\Gradienter\src\org\crazyit\sensor\MyView.java

```
public class MyView extends View
{
    // 定义水平仪仪表盘图片
    Bitmap back;
    // 定义水平仪中的气泡图标
    Bitmap bubble;
    // 定义水平仪中气泡的 X、Y 坐标
```

```
        int bubbleX, bubbleY;
        public MyView(Context context, AttributeSet attrs)
        {
            super(context, attrs);
            // 加载水平仪图片和气泡图片
            back = BitmapFactory.decodeResource(getResources()
                , R.drawable.back);
            bubble = BitmapFactory
                .decodeResource(getResources(), R.drawable.bubble);
        }
        @Override
        protected void onDraw(Canvas canvas)
        {
            super.onDraw(canvas);
            // 绘制水平仪表盘图片
            canvas.drawBitmap(back, 0, 0, null);
            // 根据气泡坐标绘制气泡
            canvas.drawBitmap(bubble, bubbleX, bubbleY, null);
        }
}
```

正如上面的程序中粗体字代码所示，该自定义 View 会根据 bubbleX、bubbleY 动态地绘制气泡的位置，而这个 bubbleX、bubbleY 就需要根据方向传感器返回的第三个角度、第二个角度来动态计算。

下面是该程序中 Activity 的代码。

程序清单：codes\15\15.3\Gradienter\src\org\crazyit\sensor\Gradienter.java

```
public class Gradienter extends Activity
        implements SensorEventListener
{
        // 定义水平仪的仪表盘
        MyView show;
        // 定义水平仪能处理的最大倾斜角，超过该角度，气泡将直接位于边界
        int MAX_ANGLE = 30;
        // 定义 Sensor 管理器
        SensorManager mSensorManager;
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            // 获取水平仪的主组件
            show = (MyView) findViewById(R.id.show);
            // 获取传感器管理服务
            mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
        }
        @Override
        public void onResume()
        {
            super.onResume();
            // 为系统的方向传感器注册监听器
            mSensorManager.registerListener(this,
                mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATION),
                SensorManager.SENSOR_DELAY_GAME);
        }
        @Override
        protected void onPause()
        {
            // 取消注册
            mSensorManager.unregisterListener(this);
            super.onPause();
```

```
}
@Override
protected void onStop()
{
      // 取消注册
      mSensorManager.unregisterListener(this);
      super.onStop();
}
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy)
{
}
@Override
public void onSensorChanged(SensorEvent event)
{
      float[] values = event.values;
      // 获取触发 event 的传感器类型
      int sensorType = event.sensor.getType();
      switch (sensorType)
      {
            case Sensor.TYPE_ORIENTATION:
                  // 获取与 Y 轴的夹角
                  float yAngle = values[1];
                  // 获取与 Z 轴的夹角
                  float zAngle = values[2];
                  // 气泡位于中间时（水平仪完全水平），气泡的 X、Y 坐标
                  int x = (show.back.getWidth() - show.bubble.getWidth()) / 2;
                  int y = (show.back.getHeight() - show.bubble.getHeight()) / 2;
                  // 如果与 Z 轴的倾斜角还在最大角度之内
                  if (Math.abs(zAngle) <= MAX_ANGLE)
                  {
                        // 根据与 Z 轴的倾斜角度计算 X 坐标的变化值（倾斜角度越大，X 坐标
                        // 变化越大）
                        int deltaX = (int) ((show.back.getWidth() - show.bubble
                              .getWidth()) / 2 * zAngle / MAX_ANGLE);
                        x += deltaX;
                  }
                  // 如果与 Z 轴的倾斜角已经大于 MAX_ANGLE，气泡应到最左边
                  else if (zAngle > MAX_ANGLE)
                  {
                        x = 0;
                  }
                  // 如果与 Z 轴的倾斜角已经小于负的 MAX_ANGLE，气泡应到最右边
                  else
                  {
                        x = show.back.getWidth() - show.bubble.getWidth();
                  }
                  // 如果与 Y 轴的倾斜角还在最大角度之内
                  if (Math.abs(yAngle) <= MAX_ANGLE)
                  {
                        // 根据与 Y 轴的倾斜角度计算 Y 坐标的变化值（倾斜角度越大，Y 坐标
                        // 变化越大）
                        int deltaY = (int) ((show.back.getHeight() - show.bubble
                              .getHeight()) / 2 * yAngle / MAX_ANGLE);
                        y += deltaY;
                  }
                  // 如果与 Y 轴的倾斜角已经大于 MAX_ANGLE，气泡应到最下边
                  else if (yAngle > MAX_ANGLE)
                  {
                        y = show.back.getHeight() - show.bubble.getHeight();
                  }
                  // 如果与 Y 轴的倾斜角已经小于负的 MAX_ANGLE，气泡应到最右边
                  else
```

```
                {
                    y = 0;
                }
                // 如果计算出来的 X、Y 坐标还位于水平仪的仪表盘内，更新水平仪的气泡坐标
                if (isContain(x, y))
                {
                    show.bubbleX = x;
                    show.bubbleY = y;
                }
                // 通知系统重回 MyView 组件
                show.postInvalidate();
                break;
            }
        }
        // 计算 x、y 点的气泡是否处于水平仪的仪表盘内
        private boolean isContain(int x, int y)
        {
            // 计算气泡的圆心坐标 X、Y
            int bubbleCx = x + show.bubble.getWidth() / 2;
            int bubbleCy = y + show.bubble.getWidth() / 2;
            // 计算水平仪仪表盘的圆心坐标 X、Y
            int backCx = show.back.getWidth() / 2;
            int backCy = show.back.getWidth() / 2;
            // 计算气泡的圆心与水平仪仪表盘的圆心之间的距离
            double distance = Math.sqrt((bubbleCx - backCx) * (bubbleCx - backCx)
                + (bubbleCy - backCy) * (bubbleCy - backCy));
            // 若两个圆心的距离小于它们的半径差，即可认为处于该点的气泡依然位于仪表盘内
            if (distance < (show.back.getWidth() - show.bubble.getWidth()) / 2)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
```

该程序的关键代码就是程序中粗体字代码部分，这些代码实现了前面介绍的设计：程序检测方向传感器返回的第二个、第三个角度值，并根据第二个角度值来计算气泡的 *Y* 坐标，并根据第三个角度值来计算 *X* 坐标；计算完成后通知系统重绘 MyView 组件即可。

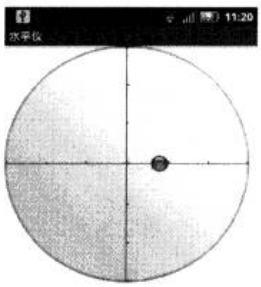


图 15.5　水平仪

如果需要在模拟器中调试该程序，请先启动 Sensor Simulator 的 PC 端，并让 SensorSimulator 的手机端程序与 PC 端程序建立连接。然后运行该程序即可看到如图 15.5 所示的结果。

从图 15.5 可以看到水平仪中间的气泡，通过该气泡所在的位置即可大致确定手机底下的支撑是否水平。当气泡位于仪表盘中的红色“十”字标识处时，即可认为手机底下的支撑完全水平。

15.4　本章小结

Android 系统的特色之一就是支持传感器，通过传感器可以程序获取手机设备运行的外界信息，包括手机运动的加速度、手机摆放方向等。学习本章需要重点掌握 Android 传感器支持的 API，包括如何通过 SensorManager 注册传感器监听器，如何使用 SensorEventListener 监听传感器数据等。除此之外，读者还需要掌握 Android 的加速度传感器、方向传感器、磁场传感器、温度传感器、光传感器和压力传感器等常见传感器的功能和用法。

CHAPTER

16

第16章 GPS应用开发

本章要点

- GPS 的概念和用途
- Android 提供的 GPS 支持
- LocationManager 和 LocationProvider
- 获取系统的 LocationProvider 列表
- 根据名字获取指定的 LocationProvider
- 根据 Criteria 获取满足条件的 LocationProvider
- 通过模拟器发送 GPS 定位信息
- 通过 LocationListener 监听 GPS 定位信息
- 临近警告支持

GPS 是英文 Global Positioning System（全球定位系统）的简称，GPS 是 20 世纪 70 年代由美国陆海空三军联合研制的新一代空间卫星导航定位系统 。从这个介绍不难发现，GPS 的作用就是为全球的物体提供定位功能。

GPS 定位系统由三部分组成，即由 GPS 卫星组成的空间部分，若干地面站组成的控制部分和普通用户手中的接收机这三个部分。对于手机用户来说，手机就是 GPS 定位系统的接收机，也就是说 GPS 定位需要手机的硬件支持 GPS 功能。

GPS 的空间部分由 GPS 卫星组成，覆盖于全球上空的 GPS 卫星星座，必须保证在各处能时时观测到高度角为 15 度以上的 4 颗卫星，这样才能保证 GPS 系统的准确定位；目前 GPS 卫星星座共有 24 颗 GPS 卫星，均匀分布在倾角为 55 度的 6 个轨道上，保证在地面的任意一点都可同时观测到 24 颗卫星。

GPS 定位系统听上去专业、高深，实际上也是一门高新技术，但对于 Android 应用开发的程序员来说，开发提供 GPS 功能的应用程序十分简单。就像 Android 为电话管理支持提供了 TelephonyManager 类、为音频管理支持提供了 AudioManager 类，Android 为支持 GPS 则提供了 LocationManager，通过 LocationManager 类及其他几个辅助类，开发人员可以非常方便地开发出 GPS 应用。

16.1　支持 GPS 的核心 API

Android 为 GPS 功能支持专门提供了一个 LocationManager 类，它的作用与 TelephonyManager、AudioManager 等服务类的作用相似，所有 GPS 定位相关的服务、对象都将由该对象来产生。

与程序中获取 TelephonyManager、AudioManager 的方法相似，程序并不能直接创建 LocationManager 的实例，而是通过调用 Context 的 getSystemService()方法来获取，例如如下代码：

```
LocationManager lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

一旦在程序中获得了 LocationManager 对象之后，接下来即可调用 LocationManager 的方法来获取 GPS 定位的相关服务和对象了。LocationManager 提供了如下常用的方法。

- boolean addGpsStatusListener(GpsStatus.Listener listener)：添加一个监听 GPS 状态的监听器。
- void addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)：添加一个临近警告。
- List<String> getAllProviders()：获取所有的 LocationProvider 列表。
- String getBestProvider(Criteria criteria, boolean enabledOnly)：根据指定条件返回最优的 LocationProvider 对象。
- GpsStatus getGpsStatus(GpsStatus status)：获取 GPS 状态。
- Location getLastKnownLocation(String provider)：根据 LocationProvider 获取最近一次已知的 Location。
- LocationProvider getProvider(String name)：根据名称来获取 LocationProvider。
- List<String> getProviders(Criteria criteria, boolean enabledOnly)：根据指定条件获取满足该条件的全部 LocationProvider 的名称。

- List<String> getProviders(boolean enabledOnly)：获取所有可用的 LocationProvider。
- boolean isProviderEnabled(String provider)：判断指定名称的 LocationProvider 是否可用。
- void removeGpsStatusListener(GpsStatus.Listener listener)：删除 GPS 状态监听器。
- void removeProximityAlert(PendingIntent intent)：删除一个临近警告。
- void requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)：通过指定的 LocationProvider 周期性地获取定位信息，并通过 intent 启动相应的组件。
- void requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)：通过指定的 LocationProvider 周期性地获取定位信息，并触发 listener 所对应的触发器。

在上面的方法列表中涉及一个 GPS 定位支持的另一个重要的 API：LocationProvider（定位提供者），LocationProvider 对象就是定位组件的抽象表示，通过 LocationProvider 可以获取该定位组件的相关信息。LocationProvider 提供了如下常用方法。

- int getAccuracy()：返回该 LocationProvider 的精度。
- String getName()：返回该 LocationProvider 的名称。
- int getPowerRequirement()：获取该 LocationProvider 的电源需求。
- boolean hasMonetaryCost()：返回该 LocationProvider 是收费的还是免费的。
- boolean meetsCriteria(Criteria criteria)：判断该 LocationProvider 是否满足 Criteria 条件。
- boolean requiresCell()：判断该 LocationProvider 是否需要访问网络基站。
- boolean requiresNetwork()：判断该 LocationProvider 是否需要网络数据。
- boolean requiresSatellite()：判断该 LocationProvider 是否需要访问基于卫星的定位系统。
- boolean supportsAltitude()：判断该 LocationProvider 是否支持高度信息。
- boolean supportsBearing()：判断该 LocationProvider 是否支持方向信息。
- boolean supportsSpeed()：判断该 LocationProvider 是否支持速度信息。

除此之外，GPS 支持还有一个支持 API：Location，它就是一个代表位置信息的抽象类，提供了如下方法来获取定位信息。

- float getAccuracy()：获取定位信息的精度。
- double getAltitude()：获取定位信息的高度。
- float getBearing()：获取定位信息的方向。
- double getLatitude()：获取定位信息的纬度。
- double getLongitude()：获取定位信息的经度。
- String getProvider()：获取提供该定位信息的 LocationProvider。
- float getSpeed()：获取定位信息的速度。
- boolean hasAccuracy()：判断该定位信息是否有精度信息。
- boolean hasAltitude()：判断该定位信息是否有高度信息。
- boolean hasBearing()：判断该定位信息是否有方向信息。

➢ boolean hasSpeed()：判断该定位信息是否有速度信息。

上面三个 API 就是 Andoid GPS 支持的三个核心 API，使用它们来获取 GPS 定位信息的通用步骤为：

（1）获取系统的 LocationManager 对象。

（2）使用 LocationManager，通过指定 LocationProvider 来获取定位信息，定位信息由 Location 对象来表示。

（3）从 Location 对象中获取定位信息。

下面将会使用这三个核心 API 进行系统定位。

16.2 获取 LocationProvider

通过前面的介绍可以看出，Android 的定位信息由 LocationProvider 对象来提供，该对象代表一个抽象的定位组件。在开始编程之前，需要先获得 LocationProvider 对象。

16.2.1 获取所有可用的 LocationProvider

LocationManager 提供了一个 getAllProviders()方法来获取系统所有可用的 LocationProvider，下面的示例程序将可以列出系统所有的 LocationProvider。该程序界面很简单，界面中只提供一个 ListView 来显示所有 LocationProvider 即可，故不再给出界面布局代码。该程序的代码如下。

程序清单：codes\16\16.2\AllProvidersTest\src\org\crazyit\gps\AllProvidersTest.java

```
public class AllProvidersTest extends Activity
{
    ListView providers;
    LocationManager lm;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        providers = (ListView) findViewById(R.id.providers);
        // 获取系统的 LocationManager 对象
        lm = (LocationManager)getSystemService(
            Context.LOCATION_SERVICE);
        // 获取系统所有的 LocationProvider 的名称
        List<String> providerNames = lm.getAllProviders();
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            this,
            android.R.layout.simple_list_item_1,
            providerNames);
        // 使用 ListView 来显示所有可用的 LocationProvider
        providers.setAdapter(adapter);
    }
}
```



图 16.1　获取系统所有的 LocationProvider

上面的程序中粗体字代码就可获取系统中所有 LocationManager 的名称。运行上面的程序，可以看到如图 16.1 所示的输出。

从图 16.1 的运行结果可以看出，当前模拟器所有可用的 Location Provider 有如下两个。

- passive：由 LocationManager.PASSIVE_PROVIDER 常量表示。
- gps：由 LocationManager.GPS_PROVIDER 常量表示。代表通过 GPS 获取定位信息的 LocationProvider 对象。

上面列出的 LocationProvider 中最常用的 LocationProviderGPS_PROVIDER。



提示：

除了上面列出的 passive 和 gps 两个 LocationProvider 之外，还有一个名为 network 的 LocationProvider，由 LocationManager.NETWORK_PROVIDER 常量表示。代表通过移动通信网络获取定位信息的 Location Provider 对象。

16.2.2 通过名称获得指定 LocationProvider

程序调用 LocationManager 的 getAllProviders()方法获取所有 LocationProvider 时返回的是 List<String>集合，集合元素为 LocationProvider 的名称，为了获取实际的 LocationProvider 对象，可借助于 LocationManager 的 LocationProvider getProvider(String name)方法。

例如以下代码：

```
// 获取基于 GPS 的 LocationProvider
LocationProvider locProvider = lm.getProvider(LocationManager.GPS_PROVIDER);
```

16.2.3 根据 Criteria 获得 LocationProvider

前面的程序调用 LocationManager 的 getAllProviders()方法返回了系统所有可用的 Location Provider，但大部分时候，应用程序可能希望得到符合指定条件的 LocationProvider，这就需要借助于 LocationManager 的 getBestProvider(Criteria criteria, boolean enabledOnly)方法来获取。

上面的方法中 Criteria 就代表了一个“过滤”条件，该方法将只返回符合该 Criteria 的 LocationProvider，Criteria 提供例如下常用的方法来设置条件。

- setAccuracy(int accuracy)：设置对 LocationProvider 的精度要求。
- setAltitudeRequired(boolean altitudeRequired)：设置要求 LocationProvider 能提供高度信息。
- setBearingRequired(boolean bearingRequired)：设置要求 LocationProvider 能提供方向信息。
- setCostAllowed(boolean costAllowed)：设置要求 LocationProvider 是否免费。
- setPowerRequirement(int level)：设置要求 LocationProvider 的耗电量。
- setSpeedRequired(boolean speedRequired)：设置要求 LocationProvider 能提供速度信息。

下面的程序示范了如何获取系统中免费的 LocationProvider，并且该 LocationProvider 必须能提供高度信息、速度信息等。

程序清单：codes\16\16.2\FreeProvidersTest\src\org\crazyit\gps\FreeProvidersTest.java

```
public class FreeProvidersTest extends Activity
{
    ListView providers;
    LocationManager lm;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        providers = (ListView) findViewById(R.id.providers);
        // 获取系统的 LocationManager 对象
        lm = (LocationManager)getSystemService(
            Context.LOCATION_SERVICE);
        // 创建一个 LocationProvider 的过滤条件
        Criteria cri = new Criteria();
        // 设置要求 LocationProvider 必须是免费的。
        cri.setCostAllowed(false);
        // 设置要求 LocationProvider 能提供高度信息
        cri.setAltitudeRequired(true);
        // 设置要求 LocationProvider 能提供方向信息
        cri.setBearingRequired(true);
        // 获取系统所有复合条件的 LocationProvider 的名称
        List<String> providerNames = lm.getProviders(cri , false);
        System.out.println(providerNames.size());
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(
            this,
            android.R.layout.simple_list_item_1,
            providerNames);
        // 使用 ListView 来显示所有可用的 LocationProvider
        providers.setAdapter(adapter);
    }
}
```

上面的程序中粗体字代码创建了一个 Criteria 对象，并通过设置了 LocationProvider 必须满足的条件，运行该程序，即可列出所有符合 Criteria 条件的 LocationProvider。

16.3　获取定位信息

获取了 LocationManager 对象之后，接下来就可通过指定 LocationProvider 获取定位信息。

➤➤16.3.1　通过模拟器发送 GPS 信息

Android 模拟器本身并不能作为 GPS 接收机，因此无法得到 GPS 的定位信息，但为了方便程序员测试 GPS 应用，Android 提供的 DDMS 工具可以发送模拟的 GPS 定位信息。

启动 Android 模拟器之后，接下来打开 DDMS 的 Emulator Control 面板即可向 Android 模拟器发送 GPS 定位信息，如图 16.2 所示。



图 16.2　向模拟器发送定位信息

在图 16.2 所示的“Emulator Control”面板中输入经度值、纬度值，然后单击“Send”按钮即可向模拟器发送 GPS 定位信息。

16.3.2 获取定位数据

下面程序示范了如何通过手机实时地获取定位信息，包括用户所在的经度、纬度、高度、方向、移动速度等。该程序的界面很简单，只提供一个文本框来显示用户的定位信息即可，故此处不再给出程序界面代码。该程序代码如下。

程序清单：codes\16\16.3\LocationTest\src\org\crazyit\gps\LocationTest.java

```
public class LocationTest extends Activity
{
    // 定义 LocationManager 对象
    LocationManager locManager;
    // 定义程序界面中的 EditText 组件
    EditText show;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取程序界面上的 EditText 组件
        show = (EditText) findViewById(R.id.show);
        // 创建 LocationManager 对象
        locManager = (LocationManager) getSystemService
            (Context.LOCATION_SERVICE);
        // 从 GPS 获取最近的最近的定位信息
        Location location = locManager.getLastKnownLocation(
            LocationManager.GPS_PROVIDER);
        // 使用 location 根据 EditText 的显示
        updateView(location);
        // 设置每 3 秒获取一次 GPS 的定位信息
        locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER
            , 3000, 8, new LocationListener()  //①
        {
            @Override
            public void onLocationChanged(Location location)
            {
                // 当 GPS 定位信息发生改变时，更新位置
                updateView(location);
            }
            @Override
            public void onProviderDisabled(String provider)
            {
                updateView(null);
            }
            @Override
            public void onProviderEnabled(String provider)
            {
                // 当 GPS LocationProvider 可用时，更新位置
                updateView(locManager
                    .getLastKnownLocation(provider));
            }
            @Override
            public void onStatusChanged(String provider, int status,
                Bundle extras)
            {
            }
        });
    }
    // 更新 EditText 中显示的内容
    public void updateView(Location newLocation)
```

```
        {
            if (newLocation != null)
            {
                StringBuilder sb = new StringBuilder();
                sb.append("实时的位置信息：\n");
                sb.append("经度：");
                sb.append(newLocation.getLongitude());
                sb.append("\n纬度：");
                sb.append(newLocation.getLatitude());
                sb.append("\n高度：");
                sb.append(newLocation.getAltitude());
                sb.append("\n速度：");
                sb.append(newLocation.getSpeed());
                sb.append("\n方向：");
                sb.append(newLocation.getBearing());
                show.setText(sb.toString());
            }
            else
            {
                // 如果传入的 Location 对象为空则清空 EditText
                show.setText("");
            }
        }
    }
```

上面的程序中粗体字代码用于从 Location 中获取定位信息，包括用户的经度、纬度、高度、方向和移动速度等信息。程序中①号粗体字代码通过 LocationManager 设置了一个监听器，该监听器负责每隔 3 秒向 LocationProvider 请求一次定位信息，当 LocationProvider 可用时、不可用时或提供的定位信息发生改变时，系统会回调 updateView(Location newLocation)来更新 EditText 中显示的定位信息。

该程序需要有访问 GPS 信号的权限，因此需要在 AndroidManifest.xml 文件中增加如下授权代码片段：

```
<!-- 授权获取定位信息 -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

运行该程序，然后通过 DDMS 的 Emulator Control 面板来发送 GPS 定位信息，即可看到该程序显示如图 16.3 所示的输出。

由于该程序每隔 3 秒就会向 GPS LocationProvider 获取一次定位信息，这样上面的程序界面上总可以实时显示该用户的定位信息。

如果把该程序与 Google Map 结合，让该程序根据 GPS 提供的信息实时地显示用户在地图上的位置，即可开发出 GPS 导航系统。下一章会介绍相关内容。



图 16.3　实时获取定位信息

16.4　临近警告

前面介绍 LocationManager 时已经提到，该 API 提供了一个 addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)方法，该方法用于添加一个临近警告。

所谓临近警告的示意如图 16.4 所示。

也就是当用户手机不断临近指定固定点时，当与该固定点的距离小于指定范围时，系统可以触发相应的处理。

添加临近警告的方法的参数的说明如下。

- latitude：指定固定点的经度。
- longitude：指定固定点的纬度。
- radius：该参数指定一个半径长度。
- expiration：该参数指定经过多少毫秒后该临近警告就会过期失效。-1 指定永不过期。
- intent：该参数指定临近该固定点时触发该 intent 对应的组件。

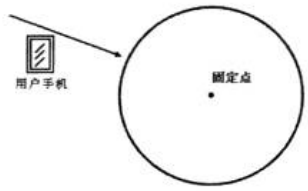


图 16.4 临近警告的示意

下面的程序示范了如何检测手机是否进入广州天河区，该程序几乎没有界面。当程序启动后，程序就会添加一个临近警告，当用户临近广州市天河区所在经度、纬度时，系统会显示提示。该程序代码如下。

程序清单：codes\16\16.4\ProximityTest\src\org\crazyit\gps\ProximityTest.java

```
public class ProximityTest extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 定位服务常量
        String locService = Context.LOCATION_SERVICE;
        // 定位服务管理器实例
        LocationManager locationManager;
        // 通过 getSystemService 方法获得 LocationManager 实例
        locationManager = (LocationManager) getSystemService(locService);
        // 定义广州天河的大致经度、纬度
        double longitude = 113.39;
        double latitude = 23.13;
        // 定义半径（5 公里）
        float radius = 5000;
        // 定义 Intent
        Intent intent = new Intent(this, ProximityAlertReciever.class);
        // 将 Intent 包装成 PendingIntent
        PendingIntent pi = PendingIntent.getBroadcast(this, -1, intent, 0);
        // 添加临近警告
        locationManager.addProximityAlert(latitude, longitude, radius, -1, pi);
    }
}
```

上面的程序中粗体字代码用于添加临近警告，当用户手机临近指定经度、纬度确定的点时，系统会启动 pi 所对应的组件。pi 对应的组件是一个 BroadcastReceiver，它的代码如下。

程序清单：codes\16\16.4\ProximityTest\src\org\crazyit\gps\ProximityAlertReciever.java

```
public class ProximityAlertReciever extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        // 获取是否为进入指定区域
        boolean isEnter = intent.getBooleanExtra(
            LocationManager.KEY_PROXIMITY_ENTERING, false);
        if(isEnter)
        {
            // 显示提示信息
```

```
                Toast.makeText(context
                        , "您已经进入广州天河区"
                        , Toast.LENGTH_LONG)
                        .show();
            }
            else
            {
                // 显示提示信息
                Toast.makeText(context
                        , "您已经离开广州天河区"
                        , Toast.LENGTH_LONG)
                        .show();
            }
        }
    }
```

该 BroadcastReceiver 被激发后的处理非常简单，程序通过 Intent 传递过来的消息判断设备是进入指定区域还是离开指定区域，并根据不同的状态提示不同的信息。

运行该程序，并通过 DDMS 的 Emulator Control 面板输入广州天河的经度、维度（113.39、23.13），即可看到如图 16.5 所示的提示。

如果接下来在 DDMS 的 Emulator Control 面板输入其他的经度、纬度，就意味着该设备离开了该区域，于是可以看到如图 16.6 所示的提示。



图 16.5　进入区域的提示



图 16.6　离开区域的提示

16.5　本章小结

本章主要介绍了 Android 提供的 GPS 支持，目前的绝大部分 Android 手机都提供了 GPS 硬件支持，都可以作为 GPS 定位系统的接收机，而开发者要做的就是从 Android 系统中获取 GPS 定位信息。学习本章的重点是掌握 LocationManager、LocationProvider 与 LocationListener 等 API 的功能和用法，并可以通过它们来监听、获取 GPS 定位信息。

一旦在应用程序中获取了 GPS 定位信息之后，接下来就可以通过这种定位信息在 Google Map 上进行定位、跟踪等，这就需要结合下一章所介绍的 Google Map 服务了。

CHAPTER

17

第17章 使用Google Map服务

本章要点

- 了解 Google Map 服务
- 掌握调用 Google Map 服务的方法
- 获取 Google Map API Key
- 创建支持 Google Map API 的 AVD
- 根据 GPS 信息在地图上定位
- 根据 GPS 信息地图上跟踪用户轨迹
- 调用 Google 的地址解析服务
- 根据地址在地图上定位

上一章介绍了如何使用 Android 的`GPS 来获取设备的定位信息，但这种方式得到的定位信息只不过是一些数字的经度、纬度值，如果这些经度、纬度值不能以更形象、直观的方式显示出来，对于大部分普通用户而言，这些经度、纬度数据几乎没有任何价值。

为了让上一章介绍的 GPS 信息“派上”用场，本章将会详细介绍 Android 提供的 Google Map 服务，Google 提供了大量的在线服务，比如 Google Map、Google 地球，Google 街景，Google 天气预报等，其实调用这些服务都比较简单，通过本章介绍的 Google Map 服务即可感受到这一点。如果把上一章获得的 GPS 信息与本章的 Map 应用结合起来，可以非常方便地开发出定位、导航等应用程序。

17.1 调用 Google Map 的准备



图 17.1 查看调试 Android 的 keystore

Android 系统默认并不支持调用 Google Map，为了正常调用 Google Map 服务，需要先进行如下准备工作。

17.1.1 获取 Map API Key

为了在应用程序中调用 Google Map，必须先获取 Google Map API 的 Key。获取步骤如下。

（1）单击 Eclipse 主菜单的“Window”菜单，再单击“Window”菜单中的“Preferences”菜单项，系统弹出如图 17.1 所示的对话框。

（2）单击图 17.1 所示对话框左边的 Android→Build 节点，即可在该对话框中看到 Android 模拟器点 keystore 的存储位置，接下来应用程序需要根据该 keystore 来生成 Goole API 的 Key。



提示：

图 17.1 所示的只是调试 Android 应用的 keystore，如果需要发布自己的 Android 应用，必须使用本公司的 keystore，此时将会按第 1 章介绍的方式来生成 keystore，那么接下来的步骤中用到 keystore 时，也应该用本公司的 keystore 的存储路径。

（3）为了生成 Google API Key，需要先用 JDK 提供的 keytool 工具查看 Android keystore 的认证指纹，启动命令行窗口输入如下命令：

```
keytool -list -v -keystore <Android keystore 的存储位置>
```

将上面命令中的<Android keystore 的存储位置>替换成图 17.1 所显示的 Android keystore 的存储位置（如果发布 Android 应用，应该使用本公司的 keystore 的存储路径）。

运行上面的命令，系统将会提示“输入 keystore 密码”，输入 Android 模拟器的 keystore 的默认密码：android，系统将会显示 Android 模拟器的 keystore 对应的认证指纹。图 17.2 显示了查看 Android keystore 的认证指纹的详细过程。



如果运行 keytool 工具时系统提示“找不到该命令”，则说明还未在 PATH 环境变量中增加%JAVA_HOME%/bin 路径，其中%JAVA_HOME%代表 JDK 的安装路径——JDK 的安装路径的 bin 子目录下应该包含 java.exe、javac.exe 和 keytool.exe 工具。





图 17.2 查看 Android keystore 的认证指纹

（4）记住图 17.2 所示的 keytool 为 Android keystore 生成的认证指纹，登录 https://developers.google.com/android/maps-api-signup 站点，系统显示如图 17.3 所示的页面。



提示：

图 17.3 所示的页面上方已经显示了使用 keytool 工具生成认证指纹的方法。

（5）在图 17.3 所示页面的文本框中输入 keytool 工具生成认证指纹，单击“Generate API Key”按钮，系统显示如图 17.4 所示的页面。

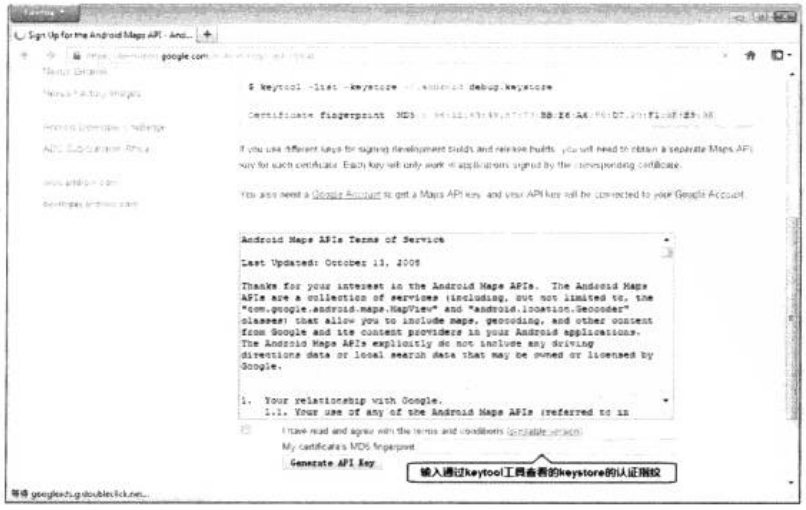


图 17.3 输入认证指纹



图 17.4　输入 Google 账户

注意：

如果打不开图 17.3 所示页面，或打开了该页面，但页面下方没有显示输入认证指纹的输入框，或打不开图 17.4 所示的页面，你不要感到奇怪，想想所处的环境就明白了！你也不要问笔者怎么解决，笔者也没有办法，笔者的方法是：让美国的朋友帮你打开页面、并根据认证指纹来获取 API Key。

（6）在图 17.4 所示页面的右边输入用户的 Google 账户，如果用户还没有 Google 账户，则可以先注册一次。如果已经有了 Google 账户，输入 Google 账户即可。登录后系统将会显示如图 17.5 所示的页面。

提示：

如果用户开始就已经登录了 Google 站点，将不会看到如图 17.4 所示的登录页面，而是直接看到图 17.5 所示的页面。图 17.5 所示页面上有些乱码，这些乱码不影响我们在 Android 应用中使用 Google Map，因此别去管它。



图 17.5　为 Google Map API 生成 Key

正如图 17.5 所示页面中看到的，页面上已经提供了如何在 Android 应用开发中使用 Google Map，页面中的 XML 代码就是使用 Google Map 的布局代码。

17.1.2　创建支持 Google Map API 的 AVD

Andriod SDK 默认并不支持 Google Map，为了得到支持 Google Map 的 SDK，必须为

Adndrid SDK 增加 Google API。启动 Android 的 SDK Manager.exe 工具，将看到如图 17.6 所示的窗口。

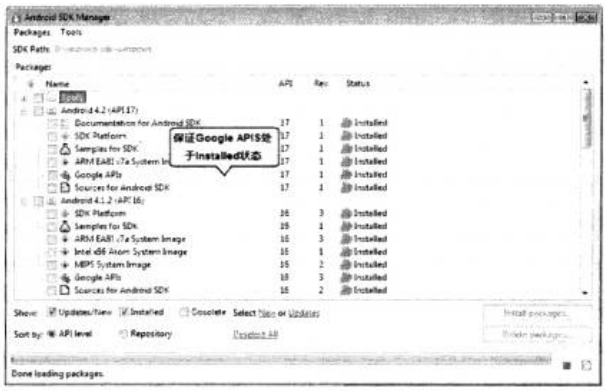


图 17.6 保证已经安装过 Google APIs

如果图 17.6 所示窗口中 Google APIs 还没有安装、还处于 Not installed 状态，则需要勾选该选项，并按第 1 章所介绍的方式安装 Google APIs。

为 Android SDK 添加了支持 Google APIs 之后，接下来还需要创建一个支持 Google APIs 的虚拟设备。创建支持 Google APIs 的虚拟设备按如下步骤进行：

① 启动 Android 的 SDK Manager.exe 工具，单击左边的“Available packages”节点，系统右边的“Virtual devices”节点，系统显示如图 17.7 所示的界面。

② 图 17.7 所示窗口中列出的虚拟设备并不支持 Google APIs，因此需要创建一个支持 Google APIs 的虚拟设备。单击“New...”按钮，系统将显示如图 17.8 所示的界面。

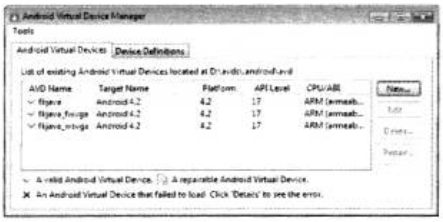


图 17.7 查看已有的虚拟设备



图 17.8 创建虚拟设备

③ 在图 17.8 所示窗口的“Target”列表中选择支持 Google Map 的平台，也就是选择 Google APIs 平台即可。

④ 剩下的设置与前面介绍的创建虚拟设备的设置完全相同，设置完成后单击“Create AVD”按钮执行创建即可。

通过上面的步骤所创建的虚拟设备可以支持运行 Android Map。

17.2 根据 GPS 信息在地图上定位

为了在 Android 平台上调用 Google Map 服务，Google Map 插件提供了一个 MapView，这个 MapView 的用法就像普通的 ImageView 一样，直接在界面布局文件中定义它，然后在程序中通过方法来控制该组件即可。

MapView 支持如下常用方法。

- MapController getController()：获取该 MapView 关联的 MapController。
- GeoPoint getMapCenter()：获取该 MapView 所显示的中心。
- int getMaxZoomLevel()：获取该 MapView 所支持的最大的放大级别。
- List<Overlay> getOverlays()：获取该 MapView 上显示的全部 Overlay。
- Projection getProjection()：获取屏幕像素坐标与经纬度坐标之间的投影关系。
- int getZoomLevel()：获取该屏幕当前的缩放级别。
- setBuiltInZoomControls(boolean on)：设置是否显示内置的缩放控制按钮。
- setSatellite(boolean on)：设置是否显示卫星地图。
- setTraffic(boolean on)：设置是否显示交通情况。

MapView 的 getController()方法会返回该 MapView 所关联的 MapController 对象，MapController 可对该 MapView 进行控制，比如控制地图定位到指定位置或控制地图放大、缩小等。

MapView 的方法中还包括一个 getOverlays()，该方法将会返回该 MapView 上所有 Overlay 对象。Overlay 是附加在 Google Map 上的附加图片，应用可以控制向 Google Map 上添加任意多个 Overlay。

为了表示 Google Map 上的指定点，Google Map 为 Android 提供了一个 GeoPoint 类。GeoPoint 十分简单，它就是对纬度、经度的封装。需要指出的是，当程序创建 GeoPoint 对象时，需要先把经度值、纬度值乘以 10 的 6 次方。除此之外，MapController 还提供了一个 animateTo(GeoPoint point)方法，该方法控制地图定位到指定的地理位置。

在 Android 应用中调用 Google Map 服务主要依赖于 MapView、MapController、GeoPoint 这三个 API，掌握了它们的用法之后，接下来就可以开发 Android 的 Map 应用了。

下面的程序示范了如何根据经度、纬度在地图上定位。该程序的界面提供了文本框让用户来输入经度、纬度。该程序的界面布局代码如下。

程序清单：codes\17\17.2\LocationMap\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<LinearLayout
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center_horizontal">
<TextView
      android:text="@string/txtLong"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
```

```
<!-- 定义输入经度值的文本框 -->
<EditText
      android:id="@+id/lng"
      android:text="@string/lng"
      android:inputType="numberDecimal"
      android:layout_width="85dp"
      android:layout_height="wrap_content" />
<TextView
      android:text="@string/txtLat"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:paddingLeft="8dp" />
<!-- 定义输入纬度值的文本框 -->
<EditText
      android:id="@+id/lat"
      android:text="@string/lat"
      android:inputType="numberDecimal"
      android:layout_width="85dp"
      android:layout_height="wrap_content" />
<Button
      android:id="@+id/loc"
      android:text="@string/loc"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="4" />
</LinearLayout>
<LinearLayout
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center_horizontal">
<!-- 定义选择地图类型的单选按钮组 -->
<RadioGroup
      android:id="@+id/rg"
      android:orientation="horizontal"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:layout_weight="1">
<RadioButton
      android:text="@string/normal"
      android:id="@+id/normal"
      android:checked="true"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<RadioButton
      android:text="@string/satellite"
      android:id="@+id/satellite"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
</RadioGroup>
</LinearLayout>
<!-- 定义一个MapView,注意apiKey必须是用户自己申请的 -->
<com.google.android.maps.MapView
      android:id="@+id/mv"
      android:clickable="true"
      android:enabled="true"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:apiKey="0eDT2SJ5RCHM_anc1QEkfLewpz7wsa3dBwrRG1Q" />
</LinearLayout>
```

上面的界面布局中定义了一个 MapView 组件，该组件就代表了 Google Map 组件。需要指出的是，该组件中 android:apiKey 必须是上一节所申请得到的 Google API Key。如上面的程序中粗体字代码所示。

注意：

使用 MapView 时候千万不要直接按书上的输入，也不要直接复制光盘中的代码，笔者所设置的 android:apiKey 是笔者自己申请得到的 Google API Key，读者需要自行填入自己的 Google API Key。



提供了上面介绍的布局之后，接下来就可以在程序中根据用户输入的经度、纬度来进行定位了。根据经度、纬度在 Google Map 上定位的步骤如下。

① 获取 MapView 对应的 MapController 对象。

② 根据程序获取的经度、纬度值创建 GeoPoint 对象。

③ 调用 MapView 所关联的 MapController 对象的 animateTo(GeoPoint point)方法定位到指定位置。

该程序代码如下。

程序清单：codes\17\17.2\LocationMap\src\org\crazyit\map\LocationMap.java

```
// 必须继承 MapActivity
public class LocationMap extends MapActivity
{
    // 定义界面上的的可视化控件
    Button locBn;
    RadioGroup mapType;
    MapView mv;
    EditText etLng , etLat;
    // 定义 MapController 对象
    MapController controller;
    Bitmap posBitmap;
    @Override
    protected void onCreate(Bundle status)
    {
        super.onCreate(status);
        setContentView(R.layout.main);
        posBitmap = BitmapFactory.decodeResource(getResources(),
            R.drawable.pos);
        // 获得界面上 MapView 对象
        mv = (MapView) findViewById(R.id.mv);
        // 获取界面上两个文本框
        etLng = (EditText) findViewById(R.id.lng);
        etLat = (EditText) findViewById(R.id.lat);
        // 设置显示放大、缩小的控制按钮
        mv.setBuiltInZoomControls(true);
        // 创建 MapController 对象
        controller = mv.getController(); //①
        // 获得 Button 对象
        locBn = (Button) findViewById(R.id.loc);
        locBn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                // 获取用户输入的经度、纬度值
```

```
                String lng = etLng.getEditableText().toString().trim();
                String lat = etLat.getEditableText().toString().trim();
                if (lng.equals("") || lat.equals(""))
                {
                    Toast.makeText(LocationMap.this, "请输入有效的经度、纬度!",
                        Toast.LENGTH_LONG).show();
                }
                else
                {
                    double dLong = Double.parseDouble(lng);
                    double dLat = Double.parseDouble(lat);
                    // 调用方法更新 MapView
                    updateMapView(dLong , dLat);
                }
            }
        });
        // 触发按钮的单击事件
        locBn.performClick();
        // 获得 RadioGroup 对象
        mapType = (RadioGroup) findViewById(R.id.rg);
        // 为 RadioGroup 的选中状态改变添加监听器
        mapType.setOnCheckedChangeListener(new OnCheckedChangeListener()
        {
            @Override
            public void onCheckedChanged(RadioGroup group, int checkedId)
            {
                switch(checkedId)
                {
                    // 如果勾选的是"正常视图"单选按钮
                    case R.id.normal:
                        mv.setSatellite(false);
                        break;
                    // 如果勾选的是"卫星视图"单选按钮
                    case R.id.satellite:
                        mv.setSatellite(true);
                        break;
                }
            }
        });
    }
    @Override
    protected boolean isRouteDisplayed()
    {
        return true;
    }
    // 根据经度、纬度将 MapView 定位到指定地点的方法
    private void updateMapView(double lng, double lat)
    {
        // 将经纬度信息包装成 GeoPoint 对象
        GeoPoint gp = new GeoPoint((int) (lat * 1E6)
            , (int) (lng * 1E6));  //②
        // 设置显示放大缩小按钮
        mv.displayZoomControls(true);
        // 将地图移动到指定的地理位置
        controller.animateTo(gp);  //③
        // 获得 MapView 上原有的 Overlay 对象
        List<Overlay> ol = mv.getOverlays();
        // 清除原有的 Overlay 对象
        ol.clear();
        // 添加一个新的 OverLay 对象
        ol.add(new PosOverLay(gp, posBitmap));  //④
```

```
        }
    }
```

上面的程序中①、②、③号粗体字代码分别用于获取 MapView 的 MapController 对象，创建 GeoPoint 对象，将地图定位到指定位置点。

当程序把地图定位到指定位置之后，程序还会在该位置“绘制”一个 Overlay 对象，用于准确标识该位置点，如以上程序④号粗体字代码所示。上面的程序添加的 Overlay 对象为 PosOverLay 实例，PosOverLay 继承了 Overlay，并在上面绘制一个简单定位图片。PosOverLay 的代码如下。

程序清单：codes\17\17.2\LocationMap\src\org\crazyit\map\PosOverLay.java

```
public class PosOverLay extends Overlay
{
    // 定义该 PosOverLay 所绘制的位图
    Bitmap posBitmap;
    // 定义该 PosOverLay 绘制位图的位置
    GeoPoint gp;
    public PosOverLay(GeoPoint gp, Bitmap posBitmap)
    {
        super();
        this.gp = gp;
        this.posBitmap = posBitmap;
    }
    @Override
    public void draw(Canvas canvas, MapView mapView
        , boolean shadow)
    {
        if (!shadow)
        {
            // 获取 MapView 的 Projection 对象
            Projection proj = mapView.getProjection();
            Point p = new Point();
            // 将真实的地理坐标转换为屏幕上的坐标
            proj.toPixels(gp, p);
            // 在指定位置绘制图片
            canvas.drawBitmap(posBitmap, p.x - posBitmap.getWidth() / 2
                , p.y - posBitmap.getHeight(), null);
        }
    }
}
```

上面的 PosOverLay 继承了 Overlay，并重写了它的 draw(Canvas canvas, MapView mapView, boolean shadow)方法。该方法负责在 Google Map 上绘制一个定位标识。通过 Overlay 在 Google Map 的指定地理位置绘制图片的需要三步。

① 获取 MapView 上屏幕坐标与经纬度坐标之间的投影关系。

② 调用 Projection 的 toPixels 方法把经纬度坐标转换为屏幕坐标。

③ 调用 Canvas 的 drawBitmap 方法在屏幕的指定位置绘制图片。

上面程序中的粗体字代码完成了上面三个步骤。

由于该程序需要使用 Google Map API，因此需要在 AndroidManifest.xml 文件的<application.../>元素中添加如下子元素：

```
<!-- 声明需要使用 Google Map API -->
<uses-library android:name="com.google.android.maps" />
```

不仅如此，使用 Google Map 时，所有的地图数据其实都来自于网络，该应用程序需要

访问网络，因此必须在 AndroidManifest.xml 文件中给该程序授予相应的权限，也就是在该文件中添加如下授权代码：

```
<!-- Google Map 需要访问互联网，所以必须授权 -->
<uses-permission android:name="android.permission.INTERNET" />
```

运行上面的程序，将看到如图 17.9 所示的结果。

如果读者开发该程序时一切正常，将可以正常看到如图 17.9 所示的地图，在程序中上方的文本框中输入合适的经度、纬度，地图将会定位到指定的位置。



图 17.9 在 Google Map 上定位

对于初学者来说，开发该程序很容易出现如下几个错误。

- com.google.android.maps it's an optionallibrary not included by default：该错误提示用户并未添加 Google Map API。也就是忘记了在 AndroidManifest.xml 文件的<application.../>元素内添加 <uses-library android:name="com.google.android.maps" />元素。
- java.lang.IllegalArgumentException: You need to specify an API Key for each MapView：该错误提示用户并未提供正确的 Google API Key。记住要将 MapView 的 android:apiKey 属性值设为用户自己申请的 API Key。
- java.lang.IllegalArgumentException: MapViews can only be created inside instances of MapActivity：该错误提示 MapView 只能在 MapActivity 中使用。也就是说，使用 Google Map 的 Activity 必须继承 MapActivity，而不是继承普通的 Activity。

17.3 GPS 导航

前面介绍的程序需要用户输入经、纬度，但实际上 Android 应用可通过 GPS 来获取定位信息。上一章已经介绍了如何通过 GPS 来获取定位信息，因此如果把前面介绍的 GPS 定位与这里的 Google Map 结合起来，可以非常方便地开发出 GPS 导航应用。

下面的应用程序开发了一个简单的 GPS 导航应用，该应用程序每隔 30 秒获取一次 GPS 定位信息，当程序得到 GPS 定位信息之后，程序就会把 Google Map 定位到该位置，这样即可在地图上实时地跟踪设备的移动位置。

下面的程序的界面布局很简单，只提供一个 MapView 来显示设备在地图上的位置即可，故此处不再给出界面布局代码。该程序的 Activity 代码如下。

程序清单：codes\17\17.3\Navigation\src\org\crazyit\map\Navigation.java

```
public class Navigation extends MapActivity
{
    MapView mv;
    MapController controller;
    Bitmap posBitmap;
    LocationManager locManager;
    @Override
    protected void onCreate(Bundle status)
    {
```

```
        super.onCreate(status);
        setContentView(R.layout.main);
        posBitmap = BitmapFactory.decodeResource(getResources(),
            R.drawable.pos);
        // 获得界面上的 MapView 对象
        mv = (MapView) findViewById(R.id.mv);
        // 设置显示放大、缩小的按钮
        mv.setBuiltInZoomControls(true);
        // 创建 MapController 对象
        controller = mv.getController();
        // 获取 LocationManager 对象
        locManager = (LocationManager) getSystemService(
            Context.LOCATION_SERVICE);
        // 设置每 30 秒获取一次 GPS 的定位信息
        locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            30000, 10, new LocationListener() //①
            {
                @Override
                public void onLocationChanged(Location location)
                {
                    // 当 GPS 定位信息发生改变时，更新位置
                    updateMapView(location); //②
                }
                @Override
                public void onProviderDisabled(String provider)
                {
                }

                @Override
                public void onProviderEnabled(String provider)
                {
                    // 当 GPS LocationProvider 可用时，更新位置
                    updateMapView(locManager
                        .getLastKnownLocation(provider));
                }
                @Override
                public void onStatusChanged(String provider, int status,
                    Bundle extras)
                {
                }
            });
    }
    @Override
    protected boolean isRouteDisplayed()
    {
        return true;
    }
    // 根据 Location 来更新 MapView
    private void updateMapView(Location location)
    {
        // 将 Location 对象中的经、纬度信息包装成 GeoPoint 对象
        GeoPoint gp = new GeoPoint((int) (location.getLatitude() * 1E6),
            (int) (location.getLongitude() * 1E6));
        // 设置显示放大、缩小按钮
        mv.displayZoomControls(true);
        // 将地图移动到指定的地理位置
        controller.animateTo(gp);
        // 获得 MapView 上原有的 Overlay 对象
        List<Overlay> ol = mv.getOverlays();
        // 清除原有的 Overlay 对象
        ol.clear();
```

```
            // 添加一个新的 OverLay 对象
            ol.add(new PosOverLay(gp, posBitmap));
        }
    }
```

上面的程序与前一个程序的区别在于：该程序每隔 30 秒向 GPS 请求一次定位数据，如程序中①号粗体字代码所示；当程序检测到 GPS 定位信息改变时，程序调用 updateMapView(Location location)方法将地图定位到指定位置，如程序中②号粗体字代码所示。

该程序需要使用 Google Map，因此需要通过互联网来获取地图信息；而且该程序还需要访问 GPS 定位信息，因此程序需要被授予访问互联网的权限和访问 GPS 信息的权限，也就是在 AndroidManifest.xml 文件中增加如下授权代码：

```
<!-- Google Map 需要访问互联网，所以必须授权 -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- 授权访问定位信息 -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

运行该程序，并通过 DDMS 的“Emulator Control”面板不断地“发送”GPS 定位信号，将可以看到该程序中地图不断加载设备的最新位置，如图 17.10 所示。

17.4 根据地址定位

前面介绍的在地图上定位的例子需要用户输入定位点的经度、纬度才能进行定位，这种方式显然不太现实：普通用户不太可能记住某个位置的经、纬度。对于普通用户来说，根据地址进行定位才是更有实用价值的地图。本节将会介绍如何根据地址在 Google Map 上定位。



图 17.10 根据 GPS 信号导航

17.4.1 地址解析与反向地址解析

由于 Google Map 的地图定位必须根据经、纬度来完成，因此如果需要让程序根据地址进行定位，则需要先把地址解析成经、纬度。这里涉及如下两个基本概念。

- 地址解析：把普通用户能看到的字符串地址转换为经、纬度。
- 反向地址解析：把经、纬度值转换成普通的字符串地址。

Android 为地址解析提供了 Geocoder 工具类，该工具类提供了如下两个方法来进行地址解析和方向地址解析。

- List<Address> getFromLocation(double latitude, double longitude, int maxResults)：执行地址解析，把经、纬度值转换为字符串地址值。
- List<Address> getFromLocationName(String locationName, int maxResults)：执行反向地址解析，把字符串地址值转换为经、纬度值。



提示：

虽然 Geocoder 工具类提供了上面两个方法来进行地址解析和反向地址解析，但实际上这个类还是需要调用网络上 Google 服务。很明显，Android 平台不大可能把地球上所有地名与经、纬度之间的映射关系都存储在手机系统中。

虽然 Android 的 API 文档中给出了 Geocoder 工具类的说明，而且也可以在程序中使用 Geocoder，在某些 Android 平台（比如 Android 2.2）上，每次在程序中调用 Geocoder 类方法时总是提示：java.io.IOException: the service is not available。幸运的是，笔者在 Android 4.2 平台上调用该工具类的方法可以正常获得结果。

如果读者对 Geocoder 工具类提供的地址解析、反向地址解析不太信任，可以借助于 Google 已公开的地址解析、反向地址解析的 API，用户登录 http://code.google.com/intl/zh-CN/apis/maps/documentation/geocoding/站点即可看到地址解析、反向地址解析 API 的详细说明。

其中地址解析的服务地址为：

```
http://maps.google.com/maps/api/geocode/json?parameters
```

反向地址解析的服务地址为：

```
http://maps.google.com/maps/api/geocode/json?latlng=40.714,-73.96&sensor=true_
or_false
```

关于 Google 地址解析、反向地址解析的 API 的用法，上面的网站上已有详细的介绍，笔者就不在这里浪费笔墨了。

由于 Google 提供了地址解析、反向地址解析的服务地址，接下来只要按如下步骤即可完成地址解析和反向地址解析。

（1）通过 HttpURLConnection 或 HttpClient 向上面任意一个地址发送请求。

（2）解析服务响应数据，获取解析结果即可。

下面的程序提供了一个 ConvertUtil 工具类，它就可以通过向上面两个地方发送请求来完成地址解析和反向地址解析。该工具类的代码如下。

程序清单：codes\17\17.4\GeocoderTest\src\org\crazyit\map\ConvertUtil.java

```
public class ConvertUtil
{
    // 根据地址获取对应的经纬度
    public static double[] getLocationInfo(final String address)
    {
        FutureTask<double[]> task = new FutureTask<double[]>(
            new Callable<double[]>()
        {
            public double[] call() throws Exception
            {
                // 定义一个 HttpClient，用于向指定地址发送请求
                HttpClient client = new DefaultHttpClient();
                // 向指定地址发送 GET 请求
                HttpGet httpGet = new HttpGet("http://maps.google.com/maps/"
                    + "api/geocode/json?address=" + address
                    + "&sensor=false");
                // 用于模拟该请求的区域来自于简体中文环境，保证返回的响应为简体中文地址
                httpGet.addHeader("Accept-Charset" , "GBK;q=0.7,*;q=0.3");
                httpGet.addHeader("Accept-Language" , "zh-CN,zh;q=0.8");
                StringBuilder sb = new StringBuilder();
                // 获取服务器的响应
                HttpResponse response = client.execute(httpGet);
                HttpEntity entity = response.getEntity();
                // 获取服务器响应的字符串
                InputStreamReader br = new InputStreamReader(
                    entity.getContent() , "utf-8");
                int b;
```

```
                while ((b = br.read()) != -1)
                {
                    sb.append((char) b);
                }
                // 将服务器返回的字符串转换为JSONObject对象
                JSONObject jsonObject = new JSONObject(sb.toString());
                // 从JSONObject对象中取出代表位置的location属性
                JSONObject location = jsonObject.getJSONArray("results")
                    .getJSONObject(0)
.getJSONObject("geometry").getJSONObject("location");
                // 获取经度信息
                double longitude = location.getDouble("lng");
                // 获取纬度信息
                double latitude = location.getDouble("lat");
                // 将经度、纬度信息组成double[]数组
                return new double[]{longitude , latitude};
            }
        });
        new Thread(task).start();
        try
        {
            return task.get();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
        return null;
    }
    // 根据经纬度获取对应的地址
    public static String getAddress(final double longitude
        , final double latitude)
    {
        FutureTask<String> task = new FutureTask<String>(
            new Callable<String>()
        {
            public String call() throws Exception
            {
                // 定义一个HttpClient，用于向指定地址发送请求
                HttpClient client = new DefaultHttpClient();
                // 向指定地址发送GET请求
                HttpGet httpGet = new HttpGet("http://maps.google.com/maps/"
                    + "api/geocode/json?latlng="
                    + latitude + "," + longitude
                    + "&sensor=false");
                // 用于模拟该请求的区域来自于简体中文环境，保证返回的响应为简体中文地址
                httpGet.addHeader("Accept-Charset" , "GBK;q=0.7,*;q=0.3");
                httpGet.addHeader("Accept-Language" , "zh-CN,zh;q=0.8");
                StringBuilder sb = new StringBuilder();
                // 执行请求
                HttpResponse response = client.execute(httpGet);
                HttpEntity entity = response.getEntity();
                // 获取服务器响应的字符串
                InputStreamReader br = new InputStreamReader(
                    entity.getContent() , "utf-8");
                int b;
                while ((b = br.read()) != -1)
                {
                    sb.append((char) b);
                }
```

```
                    // 把服务器相应的字符串转换为 JSONObject
                    JSONObject jsonObj = new JSONObject(sb.toString());
                    // 解析出响应结果中的地址数据
                    return jsonObj.getJSONArray("results").getJSONObject(0)
                        .getString("formatted_address");
                }
            });
            new Thread(task).start();
            try
            {
                return task.get();
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
            return null;
        }
    }
```

从上面的程序可以不难看出，ConvertUtil工具类的本质就是借助于HttpClient来发送GET请求，然后把服务器返回的字符串解析成JSONObject，通过JSONObject对象即可获得服务器返回的数据。

注意：

由于ConvertUtil工具类需要向指定网址发送请求才能完成地址解析、反向地址解析，因此使用该工具类的应用需要具有访问互联网的权限。



提供了上面的工具类之后，接下来的应用程序简单示范了如何利用ConvertUtil来进行地址解析、反向地址解析。该应用程序的Activity代码如下。

程序清单：codes\17\17.4\GeocoderTest\src\org\crazyit\map\GeocoderTest.java

```
public class GeocoderTest extends Activity
    implements OnClickListener
{
    Button parseBn, reverseBn;
    EditText etLng, etLat, etAddress, etResult;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 获取界面中的可视化组件
        parseBn = (Button) findViewById(R.id.parse);
        reverseBn = (Button) findViewById(R.id.reverse);
        etLng = (EditText) findViewById(R.id.lng);
        etLat = (EditText) findViewById(R.id.lat);
        etAddress = (EditText) findViewById(R.id.address);
        etResult = (EditText) findViewById(R.id.result);
        parseBn.setOnClickListener(this);
        reverseBn.setOnClickListener(this);
    }
    @Override
    public void onClick(View source)
    {
        switch (source.getId())
        {
```

```
                // 单击了“解析”按钮
                case R.id.parse:
                    String address = etAddress.getText().toString().trim();
                    if (address.equals(""))
                    {
                        Toast.makeText(this, "请输入有效的地址"
                            , Toast.LENGTH_LONG).show();
                    }
                    else
                    {
                        double[] results = ConvertUtil.getLocationInfo(address);
                        etResult.setText(address + "的经度是："
                            + results[0] + "\n纬度是："
                            + results[1]);
                    }
                    break;
                // 单击了“反向解析”按钮
                case R.id.reverse:
                    String lng = etLng.getText().toString().trim();
                    String lat = etLat.getText().toString().trim();
                    if (lng.equals("") || lat.equals(""))
                    {
                        Toast.makeText(this, "请输入有效的经度、纬度!"
                            , Toast.LENGTH_LONG)
                            .show();
                    }
                    else
                    {
                        String result = ConvertUtil.getAddress(
                            Double.parseDouble(lng), Double.parseDouble(lat));
                        etResult.setText("经度:" + lng + "、纬度:"
                            + lat + "的地址为:\n"      + result);
                    }
                    break;
            }
        }
    }
```

上面的程序中两行粗体字代码测试了 ConvertUtil 工具类的用法，分别测试了 ConvertUtil 工具类提供的地址解析与反向地址解析方法。运行上面的程序，在文本框中输入某个地名，单击“解析”按钮，将可看到如图 17.11 所示的结果。

在图 17.11 所示的输入经、纬度的文本框中输入经、纬度值，再单击“反向解析”按钮，即可看到程序有如图 17.12 所示的输出。



图 17.11 地址解析



图 17.12 反向地址解析

从图 17.13 所示的输出结果来看，程序解析 113.396、23.125 的地址为广东省广州市天河区中山大道中 85 号。需要指出的是，这里之所以可以看到简体中文地址值，是因为程序为 HttpGet 对象设置过如下两行：

```
httpGet.addHeader("Accept-Charset" , "GBK;q=0.7,*;q=0.3");
httpGet.addHeader("Accept-Language" , "zh-CN,zh;q=0.8");
```

这两行用于相当于模式简体中文的请求环境，这样保证服务器响应的数据为简体中文数据。

17.4.2 根据地址定位

下面的应用程序是对本章第一个示例程序的改进，该应用程序不需要用户输入经、纬度值，只要用户输入目标地址，程序将会调用 ConvertUtil 将字符串地址转换为经、纬度值，再控制地图定位到指定地址点即可。

该程序代码如下。

程序清单：codes\17\17.4\AddressLocMap\src\org\crazyit\map\AddressLocMap.java

```
// 必须继承 MapActivity
public class AddressLocMap extends MapActivity
{
    // 定义界面上的的可视化控件
    Button locBn;
    MapView mv;
    EditText etAddress;
    // 定义 MapController 对象
    MapController controller;
    Bitmap posBitmap;
    @Override
    protected void onCreate(Bundle status)
    {
        super.onCreate(status);
        setContentView(R.layout.main);
        posBitmap = BitmapFactory.decodeResource(getResources(),
            R.drawable.pos);
        // 获得界面上 MapView 对象
        mv = (MapView) findViewById(R.id.mv);
        etAddress = (EditText) findViewById(R.id.address);
        // 设置显示放大、缩小控制的按钮
        mv.setBuiltInZoomControls(true);
        // 创建 MapController 对象
        controller = mv.getController();
        // 获得 Button 对象
        locBn = (Button) findViewById(R.id.loc);
        locBn.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                String address = etAddress.getEditableText()
                    .toString().trim();
                if (address.equals(""))
                { // 判断是否输入空值
                    Toast.makeText(AddressLocMap.this, "请输入有效的地址!",
                        Toast.LENGTH_LONG).show();
                    return;
                }
                // 调用 ConvertUtil 执行地址解析
                double[] results = ConvertUtil.getLocationInfo(address);
                // 调用方法更新 MapView
                updateMapView(results[0], results[1]);
            }
        });
        // 触发按钮的单击事件
```

```
            locBn.performClick();
        }
        @Override
        protected boolean isRouteDisplayed()
        {
            return true;
        }
        // 根据经度、纬度将MapView定位到指定地点的方法
        private void updateMapView(double lng, double lat)
        {
            GeoPoint gp = new GeoPoint((int) (lat * 1E6)
                , (int) (lng * 1E6));
            // 设置显示放大、缩小按钮
            mv.displayZoomControls(true);
            // 将地图移动到指定的地理位置
            controller.animateTo(gp);
            // 获得MapView上原有的Overlay对象
            List<Overlay> ol = mv.getOverlays();
            // 清除原有的Overlay对象
            ol.clear();
            // 添加一个新的OverLay对象
            ol.add(new PosOverLay(gp, posBitmap));
        }
    }
```

上面的程序中粗体字代码负责把用户输入字符串地址转换成经、纬度值，然后程序调用private void updateMapView(double lng, double lat)方法把地图定位到指定位置即可。运行该程序，并在程序界面上的文本框中输入某个地址，然后单击“定位”按钮，即可看到如图17.13所示的结果。



图17.13 根据地址定位

经过前面的介绍不难发现，在Android应用中整合Google Map其实比较简单，通过为Android应用整合Google Map，可以开发出功能更强大Android应用，比如为SNS系统增加地图支持，即可让用户实时查询好友的位置等。总之，通过在Android应用中整合Google Map功能，就给开发Android应用提供了更多的可能性。

17.5 本章小结

本章主要介绍了在Android系统中调用Google Map的方法，虽然Android系统本身并未内置Google Map支持，但只要简单地选择Google API即可支持Google Map。学习本章需要掌握在Android应用中调用Google Map服务的方法，包括获取Google Map API Key、创建支持Google Map API的AVD等。除此之外，读者需要重点掌握根据GPS信息在地图上定位、跟踪的方法。不仅如此，如果结合了Android的地址解析服务，Android应用还可以根据地址信息在Google Map上定位，这也是读者需要掌握的内容。

CHAPTER

18

第18章 疯狂连连看

本章要点

- 开发单机休闲游戏的基本方法
- 单机游戏的界面分析
- 单机游戏的游戏界面与数据建模
- 开发单机游戏的界面组件
- 初始化单机游戏的状态
- 自定义 View 开发游戏主界面
- 实现游戏的 Activity
- 定义事件监听器实现游戏的人机交互
- 分情况分析游戏的逻辑处理
- 针对不同情况提供实现

本章将会介绍一个非常常见的小游戏：连连看。这个游戏是单机的休闲小游戏。连连看的游戏界面上均匀分布 2*N* 个尺寸相同的图片，每张图片在游戏中都会出现偶数次，游戏玩家需要依次找到两张相同图片，而且这两张图片之间只用横线、竖线相连（连线上不能有其他图片），并且连线的条数不超过 3 条，那么游戏会消除这两个图片。

对于 Android 学习者来说，学习开发这个小程序难度适中，而且能很好地培养学习者的学习乐趣。开发者需要从程序员的角度来看待玩家面对的游戏界面，游戏界面上的每个图片在底层只要使用一个数值标识来代表即可，不同的图片使用不同的数值表示，只要代表图片的数值相等，即可判断两张图片相同。

开发连连看游戏除了需要理解游戏界面的数据模型之外，程序开发者还需要判断两个方块是否可以相连，为了判断两个方块是否可以相连，开发者需要对两个方块所处的位置进行分类，然后针对不同的情况采用不同的判断算法进行判断，这需要开发者采用条理化的思维方式进行分析、处理，这也是学习本章需要重点掌握的能力。

18.1 连连看游戏简介

连连看是一款广受欢迎的小游戏，它具有玩法简单、耗时少等特征，尤其适合广大白领女性在办公室里休闲、放松。图 18.1 显示了连连看的游戏界面。

从图 18.1 可以看出，在连连看的游戏界面中，平均分布着 2*N* 张图片，每张图片都会出现偶数次，游戏玩家要做的事情就是依次找出两张相同的图片，如果这两张图片之间只用横线、竖线相连（连线上不能有其他图片），并且连线的条数不超过 3 条，那么游戏会消除这两个图片；当整个游戏中所有图片都被消除时，游戏结束。

图 18.2 所示左上角的两个方块之间的连接线只有 3 条，这两个方块将会被消除。



图 18.1 连连看游戏界面



图 18.2 可以消除的方块

连连看可做成单机小游戏，让脱机用户在适当的时候来独自放松；也可做成网络对战游戏——让两个用户进行比赛：谁能最先消除游戏中的所有方块，谁就胜利。

连连看的游戏界面比较简单，而且游戏的实现逻辑也不太复杂，正适合 Android 初学者作为编程进阶的练习项目。

18.2 开发游戏界面

连连看的游戏界面十分简单，大致上可分为两个区域：

- 游戏主界面区。
- 控制按钮与数据显示区。

18.2.1 开发界面布局

本程序将会使用一个 RelativeLayout 作为整体的界面布局元素，界面布局的上面是一个自定义组件，下面是一个水平排列的 LinearLayout。

程序清单：codes\18\Link\res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:background="@drawable/room">
<!-- 游戏主界面的自定义组件 -->
<org.crazyit.link.view.GameView
      android:id="@+id/gameView"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />
<!-- 水平排列的 LinearLayout -->
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:orientation="horizontal"
      android:layout_alignParentBottom="true"
      android:background="#1e72bb"
      android:gravity="center">
<!-- 控制游戏开始的按钮 -->
<Button
      android:id="@+id/startButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="@drawable/button_selector" />
<!-- 显示游戏剩余时间的文本框 -->
<TextView
      android:id="@+id/timeText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:textSize="20dp"
      android:width="150px"
      android:textColor="#ff9" />
</LinearLayout>
</RelativeLayout>
```

这个界面布局很简单，指定按钮的背景色时使用了@drawable/button_selector，这是一个在 res\drawable 目录下配置的 StateListDrawable 对象，配置文件代码如下。

程序清单：codes\18\Link\res\drawable-mdpi\button_selector.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
      <!-- 指定按钮按下时的图片 -->
      <item android:state_pressed="true"
            android:drawable="@drawable/start_down"
      />
      <!-- 指定按钮松开时的图片 -->
```

```
        <item android:state_pressed="false"
              android:drawable="@drawable/start"
        />
    </selector>
```

其中 GameView 只是一个 View 的普通子类，开发了上面的界面布局文件之后，运行该程序将可以看到如图 18.3 所示的界面。

18.2.2 开发游戏界面组件

本游戏的界面组件采用了一个自定义 View：GameView，它从 View 基类派生而出，这个自定义 View 的功能就是根据游戏状态来绘制游戏界面上的全部方块。

为了开发这个 GameView，本程序还提供了一个 Piece 类，一个 Piece 对象代表游戏界面上的一个方块，它除了封装方块上的图片之外，还需要封装该方块代表二维数组中的哪个元素；也需要封装它的左上角在游戏界面中 *X*、*Y* 坐标。图 18.4 示意了方块左上角的 *X*、*Y* 坐标的作用。



图 18.3 游戏布局



图 18.4 方块的左上角

方块左上角的 *X*、*Y* 坐标可决定它的绘制位置，GameView 根据这两个坐标值绘制全部方块即可。下面是该程序中 Piece 类的代码。

程序清单：codes\18\Link\src\org\crazyit\link\view\Piece.java

```
public class Piece
{
    // 保存方块对象的所对应的图片
    private PieceImage image;
    // 该方块的左上角的 x 坐标
    private int beginX;
    // 该方块的左上角的 y 坐标
    private int beginY;
    // 该对象在 Piece[][]数组中第一维的索引值
    private int indexX;
    // 该对象在 Piece[][]数组中第二维的索引值
    private int indexY;
    // 只设置该 Piece 对象在棋盘数组中的位置
    public Piece(int indexX , int indexY)
    {
        this.indexX = indexX;
        this.indexY = indexY;
    }
    public int getBeginX()
    {
```

```
        return beginX;
    }
    public void setBeginX(int beginX)
    {
        this.beginX = beginX;
    }
    // 下面省略了各属性的 setter 和 getter 方法
    ...
    // 判断两个 Piece 上的图片是否相同
    public boolean isSameImage(Piece other)
    {
        if (image == null)
        {
            if (other.image != null)
                return false;
        }
        // 只要 Piece 封装图片 ID 相同，即可认为两个 Piece 相等
        return image.getImageId() == other.image.getImageId();
    }
}
```

上面的 Piece 类中封装的 PieceImage 代表了该方块上的图片，但此处并未直接使用 Bitmap 对象来代表方块上的图片——因为我们需要使用 PieceImage 来封装两个信息：

- Bitmap 对象。
- 图片资源的 ID。

其中 Bitmap 对象用于在游戏界面上绘制方块；而图片资源的 ID 则代表了该 Piece 对象的标识，当两个 Piece 所封装的图片资源的 ID 相等时，即可认为这两个 Piece 上的图片相同。如以上程序中粗体字代码所示。

下面是 PieceImage 类的代码。

程序清单：codes\18\Link\src\org\crazyit\link\view\PieceImage.java

```
public class PieceImage
{
    private Bitmap image;
    private int imageId;
    // 有参数的构造器
    public PieceImage(Bitmap image, int imageId)
    {
        super();
        this.image = image;
        this.imageId = imageId;
    }
    // 省略了各属性的 setter 和 getter 方法
    ...
}
```

GameView 主要就是根据游戏的状态数据来绘制界面上的方块，GameView 继承了 View 组件，重写了 View 组件上 onDraw(Canvas canvas)方法，重写该方法主要就是绘制游戏里剩余的方块；除此之外，它还会负责绘制连接方块的连接线。

GamaView 的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\view\GameView.java

```
public class GameView extends View
{
    // 游戏逻辑的实现类
    private GameService gameService;          //①
```

```
// 保存当前已经被选中的方块
private Piece selectedPiece;
// 连接信息对象
private LinkInfo linkInfo;
private Paint paint;
// 选中标识的图片对象
private Bitmap selectImage;
public GameView(Context context, AttributeSet attrs)
{
    super(context, attrs);
    this.paint = new Paint();
    // 使用位图平铺作为连接线条
    this.paint.setShader(new BitmapShader(BitmapFactory
        .decodeResource(context.getResources(), R.drawable.heart)
        , Shader.TileMode.REPEAT, Shader.TileMode.REPEAT));
    // 设置连接线的粗细
    this.paint.setStrokeWidth(9);
    this.selectImage = ImageUtil.getSelectImage(context);
}
public void setLinkInfo(LinkInfo linkInfo)
{
    this.linkInfo = linkInfo;
}
public void setGameService(GameService gameService)
{
    this.gameService = gameService;
}
@Override
protected void onDraw(Canvas canvas)
{
    super.onDraw(canvas);
    if (this.gameService == null)
        return;
    Piece[][] pieces = gameService.getPieces();          //②
    if (pieces != null)
    {
        // 遍历 pieces 二维数组
        for (int i = 0; i < pieces.length; i++)
        {
            for (int j = 0; j < pieces[i].length; j++)
            {
                // 如果二维数组中该元素不为空（即有方块），将这个方块的图片画出来
                if (pieces[i][j] != null)
                {
                    // 得到这个 Piece 对象
                    Piece piece = pieces[i][j];
                    // 根据方块左上角 X、Y 坐标绘制方块
                    canvas.drawBitmap(piece.getImage().getImage(),
                        piece.getBeginX(), piece.getBeginY(), null);
                }
            }
        }
    }
    // 如果当前对象中有 linkInfo 对象，即连接信息
    if (this.linkInfo != null)
    {
        // 绘制连接线
        drawLine(this.linkInfo, canvas);
        // 处理完后清空 linkInfo 对象
        this.linkInfo = null;
    }
```

```
            // 画选中标识的图片
            if (this.selectedPiece != null)
            {
                canvas.drawBitmap(this.selectImage, this.selectedPiece.
                getBeginX(),
                    this.selectedPiece.getBeginY(), null);
            }
        }
        // 根据 LinkInfo 绘制连接线的方法
        private void drawLine(LinkInfo linkInfo, Canvas canvas)
        {
            // 获取 LinkInfo 中封装的所有连接点
            List<Point> points = linkInfo.getLinkPoints();
            // 依次遍历 linkInfo 中的每个连接点
            for (int i = 0; i < points.size() - 1; i++)
            {
                // 获取当前连接点与下一个连接点
                Point currentPoint = points.get(i);
                Point nextPoint = points.get(i + 1);
                // 绘制连线
                canvas.drawLine(currentPoint.x, currentPoint.y,
                    nextPoint.x, nextPoint.y, this.paint);
            }
        }
        // 设置当前选中方块的方法
        public void setSelectedPiece(Piece piece)
        {
            this.selectedPiece = piece;
        }
        // 开始游戏方法
        public void startGame()
        {
            this.gameService.start();
            this.postInvalidate();
        }
    }
```

上面的 GameView 中第一段粗体字代码用于根据游戏的状态数据来绘制界面中的所有方块，第二段粗体字代码则用于根据 LinkInfo 来绘制两个方块之间的连接线。

上面的程序中①号代码处定义了 GameService 对象，②号代码则调用了 GameService 的 getPieces()方法来获取游戏中剩余的方块，GameService 是游戏的业务逻辑实现类。后面会详细介绍该类的实现，此处暂不讲解。

18.2.3 处理方块之间的连接线

LinkInfo 是一个非常简单的工具类，它用于封装两个方块之间的连接信息——其实就是封装一个 List，List 里保存了连接线需要经过的点。



图 18.5　方块的连接

在实现 LinkInfo 对象之前，先来分析两个方块可以相连的情形。连连看游戏的规则约定：两个方块之间最多只能用 3 条线段相连，也就是说最多只能有 2 个“拐点”，加上两个方块的中心，方块的连接信息最多只需要 4 个连接点。图 18.5 显示了允许出现的连接情况。

考虑到 LinkInfo 最多需要封装 4 个连接点，最少需要封装 2 个连接点，因此程序定义如下 LinkInfo 类。

程序清单：codes\18\Link\src\org\crazyit\link\object\LinkInfo.java

```
public class LinkInfo
{
    // 创建一个集合用于保存连接点
    private List<Point> points = new ArrayList<Point>();
    // 提供第一个构造器，表示两个 Point 可以直接相连，没有转折点
    public LinkInfo(Point p1, Point p2)
    {
        // 加到集合中去
        points.add(p1);
        points.add(p2);
    }
    // 提供第二个构造器，表示三个 Point 可以相连，p2 是 p1 与 p3 之间的转折点
    public LinkInfo(Point p1, Point p2, Point p3)
    {
        points.add(p1);
        points.add(p2);
        points.add(p3);
    }
    // 提供第三个构造器，表示四个 Point 可以相连，p2，p3 是 p1 与 p4 的转折点
    public LinkInfo(Point p1, Point p2, Point p3, Point p4)
    {
        points.add(p1);
        points.add(p2);
        points.add(p3);
        points.add(p4);
    }
    // 返回连接集合
    public List<Point> getLinkPoints()
    {
        return points;
    }
}
```

LinkInfo 中所用的 Point 代表一个点，程序直接使用了 android.graphics.Point 类，每个 Point 封装了该点的 X、Y 坐标。

18.3 连连看的状态数据模型

对于游戏玩家而言，游戏界面上看到“元素”千差万别、变化多端；但对于游戏开发者而言，游戏界面上的元素在底层都是一些数据，不同数据所绘制的图片有差异而已。因此建立游戏的状态数据模型是实现游戏逻辑的重要步骤。

18.3.1 定义数据模型

连连看的游戏界面是一个 $N \times M$ 的“网格”，每个网格上显示一张图片。但对于游戏开发者来说，这个网格只需要用一个二维数据来定义即可，而每个网格上所显示的图片，对于底层的数据模型来说，不同的图片对应于不同的数值即可。图 18.6 显示了数据模型的示意。

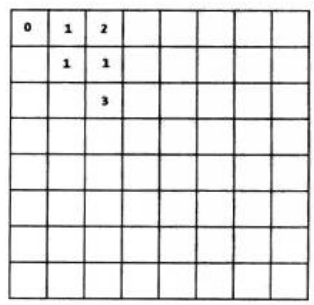


图 18.6 连连看的数据模型

对于图 18.6 所示的数据模型，只要让数值为 0 的网络上不绘制图片，其他数值的网格则绘制相应的图片，就可显示出连连看的

游戏界面了。

本程序实际上并不是直接使用 int[][]数组来保存游戏的状态数据，而是采用 Piece[][]来保存游戏的状态模型——因为 Piece 对象封装的信息更多，不仅包含了该方块的左上角的 *X*、*Y* 坐标，而且还包含了该 Piece 所显示的图片、图片 ID——这个图片 ID 就可作为该 Piece 的数据。

18.3.2 初始化游戏状态数据

为了初始化游戏状态，程序需要创建一个 Piece[][]数组，为此程序定义一个 AbstractBoard 抽象类，该抽象类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\AbstractBoard.java

```
public abstract class AbstractBoard
{
    // 定义一个抽象方法，让子类去实现
    protected abstract List<Piece> createPieces(GameConf config,
        Piece[][] pieces);
    public Piece[][] create(GameConf config)
    {
        // 创建 Piece[][]数组
        Piece[][] pieces = new Piece[config.getXSize()][config.getYSize()];
        // 返回非空的 Piece 集合，该集合由子类去创建
        List<Piece> notNullPieces = createPieces(config, pieces);     //①
        // 根据非空 Piece 对象的集合的大小来取图片
        List<PieceImage> playImages = ImageUtil.getPlayImages(config.get
        Context(),
            notNullPieces.size());
        // 所有图片的宽、高都是相同的
        int imageWidth = playImages.get(0).getImage().getWidth();
        int imageHeight = playImages.get(0).getImage().getHeight();
        // 遍历非空的 Piece 集合
        for (int i = 0; i < notNullPieces.size(); i++)
        {
            // 依次获取每个 Piece 对象
            Piece piece = notNullPieces.get(i);
            piece.setImage(playImages.get(i));
            // 计算每个方块左上角的 X、Y 坐标
            piece.setBeginX(piece.getIndexX() * imageWidth
                + config.getBeginImageX());
            piece.setBeginY(piece.getIndexY() * imageHeight
                + config.getBeginImageY());
            // 将该方块对象放入方块数组的相应位置处
            pieces[piece.getIndexX()][piece.getIndexY()] = piece;
        }
        return pieces;
    }
}
```

上面的程序中粗体字代码块用于初始化 Piece[][]数组，初始化代码负责为各非空的 Piece 元素的 beginX、beginY、image 属性赋值，其中 beginX、beginY 根据该方块在二维数组中的位置动态计算得到。

上面的程序中①号代码调用了 createPieces(config, pieces)抽象方法来创建一个 List<Piece>集合，该抽象方法将会交给其子类去实现，这里是典型的“模板模式”的应用。AbstractBoard 抽象基类完全可以根据 Piece 对象在二维数组中的位置动态地计算它的 beginX、beginY，但 AbstractBoard 不确定 Piece[][]数组的哪些元素是非空的。

由于连连看游戏的初始状态可能有很多种——比如横向分布的方块、竖向分布的方块、矩阵排列的方块、随机分布的方块等，该程序为了考虑以后的扩展性，此处只是采用了模板模式：定义 AbstractBoard 抽象基类来完成通用的代码，而暂时无法确定、需要子类实现的方法定义成 createPieces(GameConf config, Piece[][] pieces)抽象方法。

上面的程序中还用到了一个 ImageUtil 工具类，它的作用是自动搜寻/res/drawable-mdpi 目录下的图片，并根据需要随机地读取该目录下的图片。后面会详细介绍该工具类的用法。

下面为该 AbstractBoard 实现 3 个子类。

1. 矩阵排列的方块

矩阵排列的方块会填满二维数组的每个数组元素，只是把四周留空即可，该子类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\FullBoard.java

```
public class FullBoard extends AbstractBoard
{
    @Override
    protected List<Piece> createPieces(GameConf config,
        Piece[][] pieces)
    {
        // 创建一个 Piece 集合，该集合里面存放初始化游戏时所需的 Piece 对象
        List<Piece> notNullPieces = new ArrayList<Piece>();
        for (int i = 1; i < pieces.length - 1; i++)
        {
            for (int j = 1; j < pieces[i].length - 1; j++)
            {
                // 先构造一个 Piece 对象，只设置它在 Piece[][]数组中的索引值
                // 所需要的 PieceImage 由其父类负责设置
                Piece piece = new Piece(i, j);
                // 添加到 Piece 集合中
                notNullPieces.add(piece);
            }
        }
        return notNullPieces;
    }
}
```

该子类初始化的游戏界面如图 18.7 所示。



图 18.7 矩阵排列的方块

2. 竖向排列的方块

竖向排列的方块以垂直的空列分隔开，该子类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\Vertical- Board. java

```
public class VerticalBoard extends AbstractBoard
{
    protected List<Piece> createPieces(GameConf config,
        Piece[][] pieces)
    {
        // 创建一个 Piece 集合，该集合里面存放初始化游戏时所需的 Piece 对象
        List<Piece> notNullPieces = new ArrayList<Piece>();
        for (int i = 0; i < pieces.length; i++)
        {
            for (int j = 0; j < pieces[i].length; j++)
            {
```

```
                    // 加入判断，符合一定条件才去构造 Piece 对象，并加到集合中
                    if (i % 2 == 0)
                    {
                        // 如果 x 能被 2 整除，即单数列不会创建方块
                        // 先构造一个 Piece 对象，只设置它在 Piece[][]数组中的索引值
                        // 所需要的 PieceImage 由其父类负责设置
                        Piece piece = new Piece(i, j);
                        // 添加到 Piece 集合中
                        notNullPieces.add(piece);
                    }
                }
            }
            return notNullPieces;
        }
    }
```

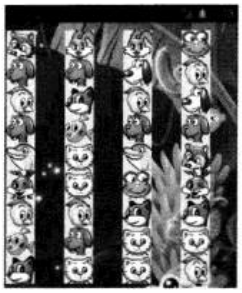


图 18.8　竖向排列的方块

上面的程序中粗体字代码控制了只设置 i % 2 == 0 的列，也就是只设置索引为偶数的列，该子类初始化的游戏界面如图 18.8 所示。

3. 横向排列的方块

竖向排列的方块以水平的空行分隔开，该子类的代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\Horizontal- Board.java

```
public class HorizontalBoard extends AbstractBoard
{
    protected List<Piece> createPieces(GameConf config,
        Piece[][] pieces)
    {
        // 创建一个 Piece 集合，该集合里面存放初始化游戏时所需的 Piece 对象
        List<Piece> notNullPieces = new ArrayList<Piece>();
        for (int i = 0; i < pieces.length; i++)
        {
            for (int j = 0; j < pieces[i].length; j++)
            {
                // 加入判断，符合一定条件才去构造 Piece 对象，并加到集合中
                if (j % 2 == 0)
                {
                    // 如果 x 能被 2 整除，即单数行不会创建方块
                    // 先构造一个 Piece 对象，只设置它在 Piece[][]数组中的索引值
                    // 所需要的 PieceImage 由其父类负责设置
                    Piece piece = new Piece(i, j);
                    // 添加到 Piece 集合中
                    notNullPieces.add(piece);
                }
            }
        }
        return notNullPieces;
    }
}
```



图 18.9　横向分布的方块

上面的程序中粗体字代码控制了只设置 j % 2 == 0 的行，也就是只设置索引为偶数的行，该子类初始化的游戏界面如图 18.9 所示。

18.4　加载界面的图片

正如前面 AbstractBoard 类的代码中看到的，当程序需要创建 *N*

个 Piece 对象时，程序会直接调用 ImageUtil 的 getPlayImages()方法去获取图片，该方法将会随机从 res\drawable-mdpi 目录下取得 *N* 张图片。

为了让 getPlayImages()方法从 res\drawable-mdpi 目录下随机取得 *N* 张图片，程序的实现思路可分为如下几步：

① 通过反射来获取 R.drawable 的所有 Field（Android 的每张图片资源都会自动转换为 R.drawable 的静态 Field），并将这些 Field 值添加到一个 List 集合中。

② 从第一步得到的 List 集合中随机“抽取” *N*/2 个图片 ID。

③ 将第二步得到的 *N*/2 个图片 ID 全部复制一份，这样就得到了 *N* 个图片 ID，而且每个图片 ID 都可以找到与之配对的。

④ 将第三步得到的 *N* 个图片 ID 再次“随机打乱”，并根据图片 ID 加载相应的 Bitmap 对象，最后把图片 ID 及对应的 Bitmap 封装成 PieceImage 后返回。

下面是 ImageUtil 类的代码。

程序清单：codes\18\Link\src\org\crazyit\link\util\ImageUtil.java

```
public class ImageUtil
{
    // 保存所有连连看图片资源值(int 类型)
    private static List<Integer> imageValues = getImageValues();
    //获取连连看所有图片的 ID（约定所有图片 ID 以 p_开头）
    public static List<Integer> getImageValues()
    {
        try
        {
            // 得到 R.drawable 所有的属性，即获取 drawable 目录下的所有图片
            Field[] drawableFields = R.drawable.class.getFields();
            List<Integer> resourceValues = new ArrayList<Integer>();
            for (Field field : drawableFields)
            {
                // 如果该 Field 的名称以 p_开头
                if (field.getName().indexOf("p_") != -1)
                {
                    resourceValues.add(field.getInt(R.drawable.class));
                }
            }
            return resourceValues;
        }
        catch (Exception e)
        {
            return null;
        }
    }
    /**
     * 随机从 sourceValues 的集合中获取 size 个图片 ID，返回结果为图片 ID 的集合
     * @param sourceValues 从中获取的集合
     * @param size 需要获取的个数
     * @return size 个图片 ID 的集合
     */
    public static List<Integer> getRandomValues(List<Integer> sourceValues,
        int size)
    {
        // 创建一个随机数生成器
        Random random = new Random();
        // 创建结果集合
        List<Integer> result = new ArrayList<Integer>();
        for (int i = 0; i < size; i++)
```

```
        {
            try
            {
                // 随机获取一个数字，大于、小于 sourceValues.size()的数值
                int index = random.nextInt(sourceValues.size());
                // 从图片 ID 集合中获取该图片对象
                Integer image = sourceValues.get(index);
                // 添加到结果集中
                result.add(image);
            }
            catch (IndexOutOfBoundsException e)
            {
                return result;
            }
        }
        return result;
    }
    /**
     * 从 drawable 目录中中获取 size 个图片资源 ID，其中 size 为游戏数量
     * @param size 需要获取的图片 ID 的数量
     * @return size 个图片 ID 的集合
     */
    public static List<Integer> getPlayValues(int size)
    {
        if (size % 2 != 0)
        {
            // 如果该数除以 2 有余数，将 size 加 1
            size += 1;
        }
        // 再从所有的图片值中随机获取 size 的一半数量
        List<Integer> playImageValues = getRandomValues(imageValues, size / 2);
        // 将 playImageValues 集合的元素增加一倍（保证所有图片都有与之配对的图片）
        playImageValues.addAll(playImageValues);
        // 将所有图片 ID 随机“洗牌”
        Collections.shuffle(playImageValues);
        return playImageValues;
    }
    /**
     * 将图片 ID 集合转换 PieceImage 对象集合，PieceImage 封装了图片 ID 与图片本身
     * @param context
     * @param resourceValues
     * @return size 个 PieceImage 对象的集合
     */
    public static List<PieceImage> getPlayImages(Context context, int size)
    {
        // 获取图片 ID 组成的集合
        List<Integer> resourceValues = getPlayValues(size);
        List<PieceImage> result = new ArrayList<PieceImage>();
        // 遍历每个图片 ID
        for (Integer value : resourceValues)
        {
            // 加载图片
            Bitmap bm = BitmapFactory.decodeResource(
                context.getResources(), value);
            // 封装图片 ID 与图片本身
            PieceImage pieceImage = new PieceImage(bm, value);
            result.add(pieceImage);
        }
        return result;
    }
    // 获取选中标识的图片
```

```
    public static Bitmap getSelectImage(Context context)
    {
        Bitmap bm = BitmapFactory.decodeResource(context.getResources(),
            R.drawable.selected);
        return bm;
    }
}
```

18.5 实现游戏 Activity

前面已经给出了游戏界面的布局文件，该布局文件需要使用一个 Activity 来负责显示，除此之外，Activity 还需要为游戏界面的按钮、GameView 组件的事件提供事件监听器。

尤其是对于 GameView 组件，程序需要监听用户的触碰动作，当用户触碰屏幕时，程序需要获取用户触碰的是哪个方块，并判断是否需要“消除”该方块。为了判断能否消除该方块，程序需要进行如下判断：

- 如果程序之前已经选中了某个方块，就判断当前触碰的方块是否能与之前的方块“相连”，如果可以相连，则消除两个方块；如果两个方块不可以相连，则把当前方块设置为选中方块。
- 如果程序之前没有选中方块，直接将当前方块设置为选中方块。

下面是该程序的 Activity 的代码。

程序清单：codes\18\Link\src\org\crazyit\link\Link.java

```
ppublic class Link extends Activity
{
    // 游戏配置对象
    private GameConf config;
    // 游戏业务逻辑接口
    private GameService gameService;
    // 游戏界面
    private GameView gameView;
    // 开始按钮
    private Button startButton;
    // 记录剩余时间的 TextView
    private TextView timeTextView;
    // 失败后弹出的对话框
    private AlertDialog.Builder lostDialog;
    // 游戏胜利后的对话框
    private AlertDialog.Builder successDialog;
    // 定时器
    private Timer timer = new Timer();
    // 记录游戏的剩余时间
    private int gameTime;
    // 记录是否处于游戏状态
    private boolean isPlaying;
    // 播放音效的 SoundPool
    SoundPool soundPool = new SoundPool(2
        , AudioManager.STREAM_SYSTEM , 8);
    int dis;
    // 记录已经选中的方块
    private Piece selected = null;
    private Handler handler = new Handler()
    {
        public void handleMessage(Message msg)
        {
```

```
            switch (msg.what)
            {
                case 0x123:
                    timeTextView.setText("剩余时间： " + gameTime);
                    gameTime--;
                    // 时间小于0，游戏失败
                    if (gameTime < 0)
                    {
                        stopTimer();
                        // 更改游戏的状态
                        isPlaying = false;
                        lostDialog.show();
                        return;
                    }
                    break;
            }
        }
    };
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        // 初始化界面
        init();
    }
    // 初始化游戏的方法
    private void init()
    {
        config = new GameConf(8, 9, 2, 10 , 100000, this);
        // 得到游戏区域对象
        gameView = (GameView) findViewById(R.id.gameView);
        // 获取显示剩余时间的文本框
        timeTextView = (TextView) findViewById(R.id.timeText);
        // 获取开始按钮
        startButton = (Button) this.findViewById(R.id.startButton);
        // 初始化音效
        dis = soundPool.load(this , R.raw.dis , 1);
        gameService = new GameServiceImpl(this.config);
        gameView.setGameService(gameService);
        // 为开始按钮的单击事件绑定事件监听器
        startButton.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View source)
            {
                startGame(GameConf.DEFAULT_TIME);
            }
        });
        // 为游戏区域的触碰事件绑定监听器
        this.gameView.setOnTouchListener(new View.OnTouchListener()
        {
            public boolean onTouch(View view, MotionEvent e)
            {
                if(!isPlaying)
                {
                    return false;
                }
                if (e.getAction() == MotionEvent.ACTION_DOWN)
                {
                    gameViewTouchDown(e);
```

```
                }
                if (e.getAction() == MotionEvent.ACTION_UP)
                {
                    gameViewTouchUp(e);
                }
                return true;
            }
        });
        // 初始化游戏失败的对话框
        lostDialog = createDialog("Lost", "游戏失败！ 重新开始", R.drawable.lost)
            .setPositiveButton("确定", new DialogInterface.OnClickListener()
            {
                public void onClick(DialogInterface dialog, int which)
                {
                    startGame(GameConf.DEFAULT_TIME);
                }
            });
        // 初始化游戏胜利的对话框
        successDialog = createDialog("Success", "游戏胜利！ 重新开始",
            R.drawable.success).setPositiveButton("确定",
            new DialogInterface.OnClickListener()
            {
                public void onClick(DialogInterface dialog, int which)
                {
                    startGame(GameConf.DEFAULT_TIME);
                }
            });
    }
    @Override
    protected void onPause()
    {
        // 暂停游戏
        stopTimer();
        super.onPause();
    }
    @Override
    protected void onResume()
    {
        // 如果处于游戏状态中
        if (isPlaying)
        {
            // 以剩余时间重写开始游戏
            startGame(gameTime);
        }
        super.onResume();
    }
    // 触碰游戏区域的处理方法
    private void gameViewTouchDown(MotionEvent event) //①
    {
        // 获取 GameServiceImpl 中的 Piece[][]数组
        Piece[][] pieces = gameService.getPieces();
        // 获取用户点击的 x 坐标
        float touchX = event.getX();
        // 获取用户点击的 y 坐标
        float touchY = event.getY();
        // 根据用户触碰的坐标得到对应的 Piece 对象
        Piece currentPiece = gameService.findPiece(touchX, touchY); //②
        // 如果没有选中任何 Piece 对象(即鼠标点击的地方没有图片)，不再往下执行
        if (currentPiece == null)
            return;
        // 将 gameView 中的选中方块设为当前方块
```

```
            this.gameView.setSelectedPiece(currentPiece);
            // 表示之前没有选中任何一个 Piece
            if (this.selected == null)
            {
                // 将当前方块设为已选中的方块，重新将 GamePanel 绘制，并不再往下执行
                this.selected = currentPiece;
                this.gameView.postInvalidate();
                return;
            }
            // 表示之前已经选择了一个
            if (this.selected != null)
            {
                // 在这里就要对 currentPiece 和 prePiece 进行判断并进行连接
                LinkInfo linkInfo = this.gameService.link(this.selected,
                    currentPiece);  //③
                // 两个 Piece 不可连，linkInfo 为 null
                if (linkInfo == null)
                {
                    // 如果连接不成功，将当前方块设为选中方块
                    this.selected = currentPiece;
                    this.gameView.postInvalidate();
                }
                else
                {
                    // 处理成功连接
                    handleSuccessLink(linkInfo, this.selected
                        , currentPiece, pieces);
                }
            }
        }
        // 触碰游戏区域的处理方法
        private void gameViewTouchUp(MotionEvent e)
        {
            this.gameView.postInvalidate();
        }

        // 以 gameTime 作为剩余时间开始或恢复游戏
        private void startGame(int gameTime)
        {
            // 如果之前的 timer 还未取消，取消 timer
            if (this.timer != null)
            {
                stopTimer();
            }
            // 重新设置游戏时间
            this.gameTime = gameTime;
            // 如果游戏剩余时间与总游戏时间相等，即为重新开始新游戏
            if(gameTime == GameConf.DEFAULT_TIME)
            {
                // 开始新的游戏
                gameView.startGame();
            }
            isPlaying = true;
            this.timer = new Timer();
            // 启动计时器，每隔 1 秒发送一次消息
            this.timer.schedule(new TimerTask()
            {
                public void run()
                {
                    handler.sendEmptyMessage(0x123);
                }
```

```
            }, 0, 1000);
            // 将选中方块设为 null。
            this.selected = null;
        }
        /**
         * 成功连接后处理
         *
         * @param linkInfo 连接信息
         * @param prePiece 前一个选中方块
         * @param currentPiece 当前选择方块
         * @param pieces 系统中还剩的全部方块
         */
        private void handleSuccessLink(LinkInfo linkInfo, Piece prePiece,
            Piece currentPiece, Piece[][] pieces)  //④
        {
            // 它们可以相连，让 GamePanel 处理 LinkInfo
            this.gameView.setLinkInfo(linkInfo);
            // 将 gameView 中的选中方块设为 null
            this.gameView.setSelectedPiece(null);
            this.gameView.postInvalidate();
            // 将两个 Piece 对象从数组中删除
            pieces[prePiece.getIndexX()][prePiece.getIndexY()] = null;
            pieces[currentPiece.getIndexX()][currentPiece.getIndexY()] = null;
            // 将选中的方块设置 null。
            this.selected = null;
            // 播放音效
            soundPool.play(dis, 1, 1, 0, 0, 1);
            // 判断是否还有剩下的方块，如果没有，游戏胜利
            if (!this.gameService.hasPieces())
            {
                // 游戏胜利
                this.successDialog.show();
                // 停止定时器
                stopTimer();
                // 更改游戏状态
                isPlaying = false;
            }
        }
        // 创建对话框的工具方法
        private AlertDialog.Builder createDialog(String title, String message,
            int imageResource)
        {
            return new AlertDialog.Builder(this).setTitle(title)
                .setMessage(message).setIcon(imageResource);
        }
        private void stopTimer()
        {
            // 停止定时器
            this.timer.cancel();
            this.timer = null;
        }
    }
```

上面的程序的 init()方法中粗体字代码负责为 gameView 组件的触碰事件绑定事件监听器，当用户触碰该区域时，事件监听器将会被触发；程序中①号粗体字代码定义的 gameViewTouchDown 方法负责处理触碰事件。它会先根据触碰点计算出触碰的方块，如②号粗体字代码所示；接下来该方法会判断是否之前已有选中的方块，如果没有，直接将当前方块设为选中方块，如果有，判断两个方块是否可以相连，如③号粗体字代码所示。

如果两个方块可以相连，程序将会从 Piece[][]数组中删除这两个方块，该逻辑由上面程

序中④号粗体字代码定义的 handleSuccessLink()方法完成。

除此之外，该程序为了控制时间流逝，定义了一个计时器，该计时器每隔 1 秒发送一条消息，程序将会根据该消息减少游戏的剩余时间。上面程序中 startGame(int gameTime)方法内的粗体字代码负责启动计时器。该消息将会交给程序的 Handler 对象处理，如上面程序中开始部分的代码。

该 Link Activity 用的两个类如下。

- GameConf：负责管理游戏的初始化设置信息。
- GameService：负责游戏的逻辑实现。

上面两个工具类中 GameConf 只是一个简单的设置类，此处不再给出介绍，读者自行参考光盘中的代码即可。

18.6 实现游戏逻辑

GameService 组件则是整个游戏逻辑实现的核心，而且 GameService 是一个可以复用的业务逻辑类，它与游戏实现平台无关，既可在 Java Swing 程序中使用，也可在 Android 游戏中使用；甚至只要稍做修改，GameService 也可移植到基于 C#平台的连连看游戏中。



备注：

征得《疯狂 Java 实战演义》作者杨恩雄的同意，本游戏的 GameService 组件基本上使用了《疯狂 Java 实战演义》一书第 7 章案例的 GameService 组件。

18.6.1 定义 GameService 组件接口

根据前面程序对 GameService 组件的依赖，程序需要 GameService 组件包含如下方法。

- start()：初始化游戏状态，开始游戏的方法。
- Piece[][] getPieces()：返回表示游戏状态的 Piece[][]数组。
- boolean hasPieces()：判断 Piece[][]数组中是否还剩 Piece 对象；如果所有 Piece 都被消除了，游戏也就胜利了。
- Piece findPiece(float touchX, float touchY)：根据触碰点的 X、Y 坐标来获取。
- LinkInfo link(Piece p1, Piece p2)：判断 p1、p2 两个方块是否可以相连。

为了考虑以后的可扩展性，先为 GameService 组件定义如下接口。

程序清单：codes\18\Link\src\org\crazyit\link\board\GameService.java

```
public interface GameService
{
    /**
     * 控制游戏开始的方法
     */
    void start();
    /**
     * 定义一个接口方法，用于返回一个二维数组
     * @return 存放方块对象的二维数组
     */
    Piece[][] getPieces();
    /**
     * 判断参数 Piece[][]数组中是否还存在非空的 Piece 对象
```

```
        * @return 如果还剩 Piece 对象则返回 true，没有则返回 false
        */
       boolean hasPieces();
       /**
        * 根据鼠标的 x 坐标和 y 坐标，查找出一个 Piece 对象
        * @param touchX 鼠标点击的 x 坐标
        * @param touchY 鼠标点击的 y 坐标
        * @return 返回对应的 Piece 对象，没有则返回 null
        */
       Piece findPiece(float touchX, float touchY);
       /**
        * 判断两个 Piece 是否可以相连，如果可以连接，则返回 LinkInfo 对象
        * @param p1 第一个 Piece 对象
        * @param p2 第二个 Piece 对象
        * @return 如果可以相连，则返回 LinkInfo 对象，如果两个 Piece 不可以连接，返回 null
        */
       LinkInfo link(Piece p1, Piece p2);
}
```

18.6.2 实现 GameService 组件

GameService 组件的前面三个方法实现起来都比较简单。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
public class GameServiceImpl implements GameService
{
      // 定义一个 Piece[][]数组，只提供 getter 方法
      private Piece[][] pieces;
      // 游戏配置对象
      private GameConf config;
      public GameServiceImpl(GameConf config)
      {
            // 将游戏的配置对象设置到本类中
            this.config = config;
      }
      @Override
      public void start()
      {
            // 定义一个 AbstractBoard 对象
            AbstractBoard board = null;
            Random random = new Random();
            // 获取一个随机数，可取值 0、1、2、3 四值
            int index = random.nextInt(4);
            // 随机生成 AbstractBoard 的子类实例
            switch (index)
            {
                  case 0:
                        // 0 返回 VerticalBoard(竖向)
                        board = new VerticalBoard();
                        break;
                  case 1:
                        // 1 返回 HorizontalBoard(横向)
                        board = new HorizontalBoard();
                        break;
                  default:
                        // 默认返回 FullBoard
                        board = new FullBoard();
                        break;
            }
            // 初始化 Piece[][]数组
```

```
            this.pieces = board.create(config);
        }
        // 直接返回本对象的 Piece[][]数组
        @Override
        public Piece[][] getPieces()
        {
            return this.pieces;
        }
        // 实现接口的 hasPieces 方法
        @Override
        public boolean hasPieces()
        {
            // 遍历 Piece[][]数组的每个元素
            for (int i = 0; i < pieces.length; i++)
            {
                for (int j = 0; j < pieces[i].length; j++)
                {
                    // 只要任意一个数组元素不为 null，也就是还剩有非空的 Piece 对象
                    if (pieces[i][j] != null)
                    {
                        return true;
                    }
                }
            }
            return false;
        }
        ...
    }
```

上面三个方法的实现都很简单，相信读者很容易理解。下面详细介绍剩下的两个方法的实现。

18.6.3 获取触碰点的方块

当用户触碰游戏界面时，事件监听器获取的是该触碰点在游戏界面上的 *X*、*Y* 坐标，但程序需要获取用户触碰的到底是哪个方块，因此程序必须把界面上的 *X*、*Y* 坐标换算成在 Piece[][]二维数组中的两个索引值。

考虑到游戏界面上每个方块的宽度、高度都是相同的，因此想将界面上的 *X*、*Y* 坐标换算成 Piece[][]二维数组中的索引也比较简单，只要拿 *X*、*Y* 坐标值除以图片的宽、高即可。下面的方法是根据触碰点 *X*、*Y* 坐标获取对应方块的代码：

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
// 根据触碰点的位置查找相应的方块
@Override
public Piece findPiece(float touchX, float touchY)
{
    // 由于在创建 Piece 对象的时候，将每个 Piece 的开始坐标加了
    // GameConf 中设置的 beginImageX/beginImageY 值，因此这里要减去这个值
    int relativeX = (int) touchX - this.config.getBeginImageX();
    int relativeY = (int) touchY - this.config.getBeginImageY();
    // 如果鼠标点击的地方比 board 中第一张图片的开始 x 坐标和开始 y 坐标要小，即没有找到相应的方块
    if (relativeX < 0 || relativeY < 0)
    {
        return null;
    }
    // 获取 relativeX 坐标在 Piece[][]数组中的第一维的索引值
    // 第二个参数为每张图片的宽
```

```
        int indexX = getIndex(relativeX, GameConf.PIECE_WIDTH);
        // 获取 relativeY 坐标在 Piece[][]数组中的第二维的索引值
        // 第二个参数为每张图片的高
        int indexY = getIndex(relativeY, GameConf.PIECE_HEIGHT);
        // 这两个索引比数组的最小索引还小，返回 null
        if (indexX < 0 || indexY < 0)
        {
            return null;
        }
        // 这两个索引比数组的最大索引还大(或者等于)，返回 null
        if (indexX >= this.config.getXSize()
            || indexY >= this.config.getYSize())
        {
            return null;
        }
        // 返回 Piece[][]数组的指定元素
        return this.pieces[indexX][indexY];
    }
```

上面的方法中两行粗体字代码用于根据触碰点 *X*、*Y* 坐标来计算它在 Piece[][]数组中的索引值。该方法调用了 getIndex(int relative , int size)进行计算。

getIndex(int relative , int size)方法的实现就是拿 relative 除以 size，只是程序需要判断可以整除和不能整除两种情况：如果可以整除，说明还在前一个方块内；如果不能整除，则对应于下一个方块。下面是 getIndex(int relative , int size)方法的代码。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
// 工具方法，根据 relative 坐标计算相对于 Piece[][]数组的第一维
// 或第二维的索引值 ，size 为每张图片边的长或者宽
private int getIndex(int relative, int size)
{
    // 表示坐标 relative 不在该数组中
    int index = -1;
    // 让坐标除以边长，没有余数，索引减 1
    // 例如点了 x 坐标为 20，边宽为 10，20 % 10 没有余数
    // index 为 1，即在数组中的索引为 1(第二个元素)
    if (relative % size == 0)
    {
        index = relative / size - 1;
    }
    else
    {
        // 有余数，例如点了 x 坐标为 21，边宽为 10，21 % 10 有余数，index 为 2
        // 即在数组中的索引为 2(第三个元素)
        index = relative / size;
    }
    return index;
}
```

18.6.4 判断两个方块是否可以相连

判断两个方块是否可以相连是本程序需要处理的最烦琐的地方：因为两个方块可以相连的情形比较多，大致可分为：

- 两个方块位于同一条水平线，可以直接相连。
- 两个方块位于同一条竖直线，可以直接相连。
- 两个方块以两条线段相连，也就是有一个拐角。

➢ 两个方块以三条线段相连，也就是有两个拐角。

下面 link(Piece p1, Piece p2)方法把这四种情况分开进行处理，代码如下。

程序清单：codes\18\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
// 实现接口的 link 方法
@Override
public LinkInfo link(Piece p1, Piece p2)
{
    // 两个 Piece 是同一个，即选中了同一个方块，返回 null
    if (p1.equals(p2))
        return null;
    // 如果 p1 的图片与 p2 的图片不相同，则返回 null
    if (!p1.isSameImage(p2))
        return null;
    // 如果 p2 在 p1 的左边，则需要重新执行本方法，两个参数互换
    if (p2.getIndexX() < p1.getIndexX())
        return link(p2, p1);
    // 获取 p1 的中心点
    Point p1Point = p1.getCenter();
    // 获取 p2 的中心点
    Point p2Point = p2.getCenter();
    // 如果两个 Piece 在同一行
    if (p1.getIndexY() == p2.getIndexY())        //①
    {
        // 它们在同一行并可以相连
        if (!isXBlock(p1Point, p2Point, GameConf.PIECE_WIDTH))
        {
            return new LinkInfo(p1Point, p2Point);
        }
    }
    // 如果两个 Piece 在同一列
    if (p1.getIndexX() == p2.getIndexX())        //②
    {
        if (!isYBlock(p1Point, p2Point, GameConf.PIECE_HEIGHT))
        {
            // 它们之间没有真接障碍，没有转折点
            return new LinkInfo(p1Point, p2Point);
        }
    }
    // 有一个转折点的情况
    // 获取两个点的直角相连的点，即只有一个转折点
    Point cornerPoint = getCornerPoint(p1Point, p2Point,
        GameConf.PIECE_WIDTH, GameConf.PIECE_HEIGHT);    //③
    if (cornerPoint != null)
    {
        return new LinkInfo(p1Point, cornerPoint, p2Point);
    }
    // 该 map 的 key 存放第一个转折点，value 存放第二个转折点
    // map 的 size()说明有多少种可以连的方式
    Map<Point, Point> turns = getLinkPoints(p1Point, p2Point,    //④
        GameConf.PIECE_WIDTH, GameConf.PIECE_WIDTH);
    if (turns.size() != 0)
    {
        return getShortcut(p1Point, p2Point, turns,
            getDistance(p1Point, p2Point));
    }
    return null;
}
```

上面的程序中 4 条粗体字代码分别代表了两个方块位于同一个水平线可直接相连，两个

方块位于同一个竖直线可直接相连，两个方块需要两条线相连，两个方块需要 3 条线相连 4 种情况。上面的方法分别考虑了这四种情况，但程序还需要为这 4 个方法提供实现。

为了实现上面 4 个方法，可以对两个 Piece 的位置关系进行归纳。

- p1 与 p2 在同一行（indexY 值相同）。
- p1 与 p2 在同一列（indexX 值相同）。
- p2 在 p1 的右上角（p2 的 indexX > p1 的 indexX，p2 的 indexY < p1 的 indexY）。
- p2 在 p1 的右下角（p2 的 indexX > p1 的 indexX，p2 的 indexY > p1 的 indexY）。

至于 p2 在 p1 的左上角，以及 p2 在 p1 的左下角这两种情况，程序可以重新执行 link 方法，将 p1 和 p2 两个参数的位置互换即可。

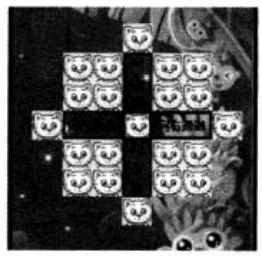


图 18.10 方块四周的通道

18.6.5 定义获取通道的工具方法

这里所谓的通道，指的是一个方块上、下、左、右四个方向上的空白方块，图 18.10 显示了一个方块四周的通道。

下面是获取某个坐标点四周通道的 4 个方法。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 给一个 Point 对象,返回它的左边通道
* @param p
 * @param pieceWidth piece 图片的宽
 * @param min 向左遍历时最小的界限
 * @return 给定 Point 左边的通道
 */
private List<Point> getLeftChanel(Point p, int min, int pieceWidth)
{
    List<Point> result = new ArrayList<Point>();
    // 获取向左通道，由一个点向左遍历，步长为 Piece 图片的宽
    for (int i = p.x - pieceWidth; i >= min
        ; i = i - pieceWidth)
    {
        // 遇到障碍，表示通道已经到尽头，直接返回
        if (hasPiece(i, p.y))
        {
            return result;
        }
        result.add(new Point(i, p.y));
    }
    return result;
}

/**
 * 给一个 Point 对象，返回它的右边通道
* @param p
 * @param pieceWidth
 * @param max 向右时的最右界限
 * @return 给定 Point 右边的通道
 */
private List<Point> getRightChanel(Point p, int max, int pieceWidth)
{
    List<Point> result = new ArrayList<Point>();
    // 获取向右通道，由一个点向右遍历，步长为 Piece 图片的宽
    for (int i = p.x + pieceWidth; i <= max
        ; i = i + pieceWidth)
```

```
        {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(i, p.y))
            {
                return result;
            }
            result.add(new Point(i, p.y));
        }
        return result;
    }

    /**
     * 给一个 Point 对象，返回它的上面通道
     * @param p
     * @param min 向上遍历时最小的界限
     * @param pieceHeight
     * @return 给定 Point 上面的通道
     */
    private List<Point> getUpChanel(Point p, int min, int pieceHeight)
    {
        List<Point> result = new ArrayList<Point>();
        // 获取向上通道，由一个点向右遍历，步长为 Piece 图片的高
        for (int i = p.y - pieceHeight; i >= min
            ; i = i - pieceHeight)
        {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(p.x, i))
            {
                // 如果遇到障碍，直接返回
                return result;
            }
            result.add(new Point(p.x, i));
        }
        return result;
    }

    /**
     * 给一个 Point 对象，返回它的下面通道
     * @param p
     * @param max 向上遍历时的最大界限
     * @return 给定 Point 下面的通道
     */
    private List<Point> getDownChanel(Point p, int max, int pieceHeight)
    {
        List<Point> result = new ArrayList<Point>();
        // 获取向下通道，由一个点向右遍历，步长为 Piece 图片的高
        for (int i = p.y + pieceHeight; i <= max
            ; i = i + pieceHeight)
        {
            // 遇到障碍，表示通道已经到尽头，直接返回
            if (hasPiece(p.x, i))
            {
                // 如果遇到障碍，直接返回
                return result;
            }
            result.add(new Point(p.x, i));
        }
        return result;
    }
```

18.6.6 没有转折点的横向连接

如果两个 Piece 的在 Piece[][]数组中的第二维索引值相等，那么这两个 Piece 就位于同一行，如前面的 link(Piece p1, Piece p2)方法中①号代码所示，此时程序需要调用 isXBlock(Point p1, Point p2, int pieceWidth)判断 p1、p2 之间是否有障碍。下面是 isXBlock 方法的代码。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 判断两个 Y 坐标相同的点对象之间是否有障碍，以 p1 为中心向右遍历
 * @param p1
 * @param p2
 * @param pieceWidth
 * @return 两个 Piece 之间有障碍返回 true，否则返回 false
 */
private boolean isXBlock(Point p1, Point p2, int pieceWidth)
{
    if (p2.x < p1.x)
    {
        // 如果 p2 在 p1 左边，调换参数位置调用本方法
        return isXBlock(p2, p1, pieceWidth);
    }
    for (int i = p1.x + pieceWidth; i < p2.x; i = i + pieceWidth)
    {
        if (hasPiece(i, p1.y))
        {// 有障碍
            return true;
        }
    }
    return false;
}
```

从上面的判断可以看出，如果两个方块位于同一行，且它们之间没有障碍，那么这两个方块就可以消除，两个方块的连接信息就是它们的中心。

18.6.7 没有转折点的纵向连接

与之相似的是，如果两个 Piece 在 Piece[][]数组中的第一维索引值相等，那么这两个 Piece 就位于同一列，如前面的 link(Piece p1, Piece p2)方法中②号代码所示，此时程序需要调用 isYBlock(Point p1, Point p2, int pieceWidth)判断 p1、p2 之间是否有障碍。下面是 isYBlock 方法的代码。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 判断两个 X 坐标相同的点对象之间是否有障碍，以 p1 为中心向下遍历
 * @param p1
 * @param p2
 * @param pieceHeight
 * @return 两个 Piece 之间有障碍返回 true，否则返回 false
 */
private boolean isYBlock(Point p1, Point p2, int pieceHeight)
{
    if (p2.y < p1.y)
    {
        // 如果 p2 在 p1 的上面，调换参数位置重新调用本方法
        return isYBlock(p2, p1, pieceHeight);
```

```
        }
        for (int i = p1.y + pieceHeight; i < p2.y; i = i + pieceHeight)
        {
            if (hasPiece(p1.x, i))
            {
                // 有障碍
                return true;
            }
        }
        return false;
    }
```

18.6.8 一个转折点的连接

对于两个方块连接线上只有一个转折点的情况，程序需要先找到这个转折点。为了找到这个转折点，程序定义一个遍历两个通道并获取它们交点的方法。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 遍历两个通道，获取它们的交点
 * @param p1Chanel 第一个点的通道
 * @param p2Chanel 第二个点的通道
 * @return 两个通道有交点，返回交点，否则返回 null
 */
private Point getWrapPoint(List<Point> p1Chanel, List<Point> p2Chanel)
{
    for (int i = 0; i < p1Chanel.size(); i++)
    {
        Point temp1 = p1Chanel.get(i);
        for (int j = 0; j < p2Chanel.size(); j++)
        {
            Point temp2 = p2Chanel.get(j);
            if (temp1.equals(temp2))
            {
                // 如果两个 List 中有元素有同一个，表明这两个通道有交点
                return temp1;
            }
        }
    }
    return null;
}
```

为了找出两个方块连接线上的连接点，程序同样需要分析 p1、p2 两个点的位置分布。根据前面的分析，我们知道 p2 要么位于 p1 的右上角，要么位于 p1 的右下角。

注意：

对于 p2 位于 p1 的左上角、左下角的情况，只要把 p1、p2 交换即可。



对于 p2 位于 p1 的右上角的情形，如图 18.11 所示。

从图 18.11 可以看出：当 p2 位于 p1 的右上角时，应该计算 p1 的左通道与 p2 的向下通道是否有交点，p1 的向上通道与 p2 的向左通道是否有交点。

对于 p2 位于 p1 的右下角的情形，如图 18.12 所示。

从图 18.12 可以看出：当 p2 位于 p1 的右上角时，应该计算 p1 的向右通道与 p2 的向上通道是否有交点，p1 的向下通道与 p2 的向左通道是否有交点。



图 18.11 p2 位于 p1 的右上角

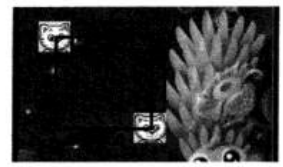


图 18.12 p2 位于 p1 的右下角

考虑到 p1 与 p2 具有上面两种分布情形，程序提供了如下方法进行处理。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 获取两个不在同一行或者同一列的坐标点的直角连接点，即只有一个转折点
 * @param point1 第一个点
 * @param point2 第二个点
 * @return 两个不在同一行或者同一列的坐标点的直角连接点
 */
private Point getCornerPoint(Point point1, Point point2, int pieceWidth,
    int pieceHeight)
{
    // 先判断这两个点的位置关系
    // point2 在 point1 的左上角，point2 在 point1 的左下角
    if (isLeftUp(point1, point2) || isLeftDown(point1, point2))
    {
        // 参数换位，重新调用本方法
        return getCornerPoint(point2, point1, pieceWidth, pieceHeight);
    }
    // 获取 p1 向右，向上，向下的三个通道
    List<Point> point1RightChanel = getRightChanel(point1, point2.x,
        pieceWidth);
    List<Point> point1UpChanel = getUpChanel(point1, point2.y, pieceHeight);
    List<Point> point1DownChanel = getDownChanel(point1, point2.y,
        pieceHeight);
    // 获取 p2 向下，向左，向下的三个通道
    List<Point> point2DownChanel = getDownChanel(point2, point1.y,
        pieceHeight);
    List<Point> point2LeftChanel = getLeftChanel(point2, point1.x,
        pieceWidth);
    List<Point> point2UpChanel = getUpChanel(point2, point1.y, pieceHeight);
    if (isRightUp(point1, point2))
    {
        // point2 在 point1 的右上角
        // 获取 p1 向右和 p2 向下的交点
        Point linkPoint1 = getWrapPoint(point1RightChanel, point2DownChanel);
        // 获取 p1 向上和 p2 向左的交点
        Point linkPoint2 = getWrapPoint(point1UpChanel, point2LeftChanel);
        // 返回其中一个交点，如果没有交点，则返回 null
        return (linkPoint1 == null) ? linkPoint2 : linkPoint1;
    }
    if (isRightDown(point1, point2))
    {
        // point2 在 point1 的右下角
        // 获取 p1 向下和 p2 向左的交点
        Point linkPoint1 = getWrapPoint(point1DownChanel, point2LeftChanel);
        // 获取 p1 向右和 p2 向下的交点
        Point linkPoint2 = getWrapPoint(point1RightChanel, point2UpChanel);
        return (linkPoint1 == null) ? linkPoint2 : linkPoint1;
    }
    return null;
}
```

上面的两行粗体字代码分别处理了 p2 位于 p1 的右上、右下的两种情形。

上面的程序中用到了 isLeftUp、isLeftDown、isRightUp、isRightDown 四个方法来判断 p2 位于 p1 的左上、左下、右上、右下 4 种情形，这 4 个方法的实现比较简单，只要对它们的 *X*、*Y* 坐标进行简单判断即可。四个方法的代码如下：

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 判断 point2 是否在 point1 的左上角
 * @param point1
 * @param point2
 * @return p2 位于 p1 的左上角时返回 true，否则返回 false
 */
private boolean isLeftUp(Point point1, Point point2)
{
    return (point2.x < point1.x && point2.y < point1.y);
}
/**
 * 判断 point2 是否在 point1 的左下角
 * @param point1
 * @param point2
 * @return p2 位于 p1 的左下角时返回 true，否则返回 false
 */
private boolean isLeftDown(Point point1, Point point2)
{
    return (point2.x < point1.x && point2.y > point1.y);
}
/**
 * 判断 point2 是否在 point1 的右上角
 * @param point1
 * @param point2
 * @return p2 位于 p1 的右上角时返回 true，否则返回 false
 */
private boolean isRightUp(Point point1, Point point2)
{
    return (point2.x > point1.x && point2.y < point1.y);
}
/**
 * 判断 point2 是否在 point1 的右下角
 * @param point1
 * @param point2
 * @return p2 位于 p1 的右下角时返回 true，否则返回 false
 */
private boolean isRightDown(Point point1, Point point2)
{
    return (point2.x > point1.x && point2.y > point1.y);
}
```

18.6.9　两个转折点的连接

两个转折点的连接又是最复杂的一种连接情况，因为两个转折点又可分为如下几种情况。

- p1、p2 位于同一行，不能直接相连，就必须有两个转折点，分向上与向下两种连接情况。
- p1、p2 位于同一列，不能直接相连，也必须有两个转折点，分向左与向右两种连接情况。
- p2 在 p1 的右下角，有 6 种转折情况。

- p2 在 p1 的右上角，同样有 6 种转折情况。

注意：

对于 p2 位于 p1 的左上角、左下角的情况，程序同样只要把 p1、p2 的位置互换即可。

对于上面 4 种情况，同样需要分别进行处理。

1. 同一行不能直接相连

p1、p2 位于同一行，但它们不能直接相连，因此必须有两个转折点，图 18.13 显示了这种相连的示意。

从图 18.13 可以看到，当 p1 与 p2 位于同一行不能直接相连时，这两个点既可在上面相连，也可在下面相连，这两种情况都代表它们可以相连，我们先把这两种情况都加入结果中，最后再去计算最近的距离。

实现时可以先构建一个 Map，Map 的 key 为第一个转折点，Map 的 value 为第二个转折点（每种连接情况最多只有两个连接点），如 Map 的 size()大于 1，说明这两个 Point 有多种连接途径，那么程序还需要计算路径最小的连接方式。

2. 同一列不能直接相连

p1、p2 位于同一列，但它们不能直接相连，因此必须有两个转折点，图 18.14 显示了这种相连的示意。

从图 18.14 可以看到，当 p1 与 p2 位于同一列不能直接相连时，这两个点既可在左边相连，也可在右边相连，这两种情况都代表它们可以相连，我们先把这两种情况都加入结果中，最后再去计算最近的距离。

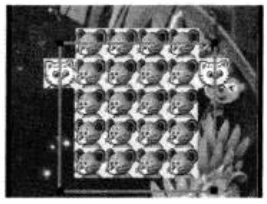


图 18.13 同一行不能直接相连

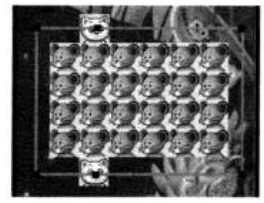


图 18.14 同一列不能直接相连

实现时可以先构建一个 Map，Map 的 key 为第一个转折点，Map 的 value 为第二个转折点（每种连接情况最多只有两个连接点），如 Map 的 size()大于 1，说明这两个 Point 有多种连接途径，那么程序还需要计算路径最小的连接方式。

3. p2 位于 p1 右下角的六种转折情况

p2 位于 p1 右下角时一共可能出现 6 种连接情况，图 18.15～图 18.20 分别绘制了这 6 种连接情况。

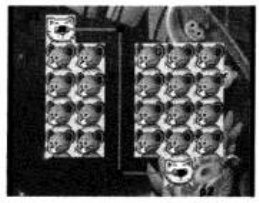


图 18.15 p2 位于 p1 右下角、两个转折点情况 1

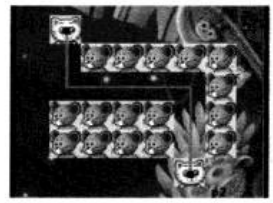


图 18.16 p2 位于 p1 右下角、两个转折点情况 2

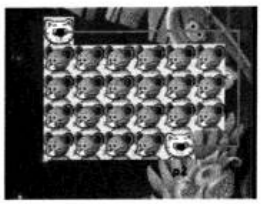


图 18.17　p2 位于 p1 右下角、两个转折点情况 3

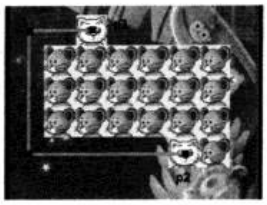


图 18.18　p2 位于 p1 右下角、两个转折点情况 4

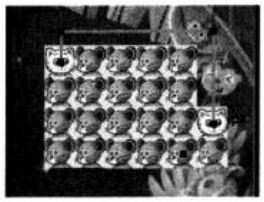


图 18.19　p2 位于 p1 右下角、两个转折点情况 5

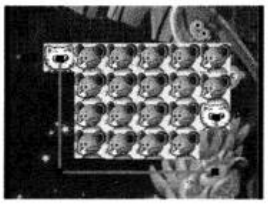


图 18.20　p2 位于 p1 右下角、两个转折点情况 6

图 18.15～图 18.20 显示了 p2 位于 p1 右下角可能出现的 6 种连接情形；实际上 p2 还可能位于 p1 的右上角，此处可能出现的 6 种连接情形也与此相似，此处不再详述。

接下来定义一个 getLinkPoints 方法对具有两个连接点的情况进行处理。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 获取两个转折点的情况
 *
 * @param point1
 * @param point2
 * @return Map 对象的每个 key-value 对代表一种连接方式，
 *   其中 key、value 分别代表第 1 个、第 2 个连接点
 */
private Map<Point, Point> getLinkPoints(Point point1, Point point2,
      int pieceWidth, int pieceHeight)
{
      Map<Point, Point> result = new HashMap<Point, Point>();
      // 获取以 point1 为中心的向上，向右，向下的通道
      List<Point> p1UpChanel = getUpChanel(point1, point2.y, pieceHeight);
      List<Point> p1RightChanel = getRightChanel(point1, point2.x, pieceWidth);
      List<Point> p1DownChanel = getDownChanel(point1, point2.y, pieceHeight);
      // 获取以 point2 为中心的向下，向左，向上的通道
      List<Point> p2DownChanel = getDownChanel(point2, point1.y, pieceHeight);
      List<Point> p2LeftChanel = getLeftChanel(point2, point1.x, pieceWidth);
      List<Point> p2UpChanel = getUpChanel(point2, point1.y, pieceHeight);
      // 获取 Board 的最大高度
      int heightMax = (this.config.getYSize() + 1) * pieceHeight
            + this.config.getBeginImageY();
      // 获取 Board 的最大宽度
      int widthMax = (this.config.getXSize() + 1) * pieceWidth
            + this.config.getBeginImageX();
      // 先确定两个点的关系
      // point2 在 point1 的左上角或者左下角
      if (isLeftUp(point1, point2) || isLeftDown(point1, point2))
      {
            // 参数换位，调用本方法
            return getLinkPoints(point2, point1, pieceWidth, pieceHeight);
      }
      // p1、p2 位于同一行不能直接相连
      if (point1.y == point2.y)
      {
```

```
        // 在同一行
        // 向上遍历
        // 以 p1 的中心点向上遍历获取点集合
        p1UpChanel = getUpChanel(point1, 0, pieceHeight);
        // 以 p2 的中心点向上遍历获取点集合
        p2UpChanel = getUpChanel(point2, 0, pieceHeight);
        Map<Point, Point> upLinkPoints = getXLinkPoints(p1UpChanel,
            p2UpChanel, pieceHeight);
        // 向下遍历，不超过 Board(有方块的地方)的边框
        // 以 p1 中心点向下遍历获取点集合
        p1DownChanel = getDownChanel(point1, heightMax, pieceHeight);
        // 以 p2 中心点向下遍历获取点集合
        p2DownChanel = getDownChanel(point2, heightMax, pieceHeight);
        Map<Point, Point> downLinkPoints = getXLinkPoints(p1DownChanel,
            p2DownChanel, pieceHeight);
        result.putAll(upLinkPoints);
        result.putAll(downLinkPoints);
    }
    // p1、p2 位于同一列不能直接相连
    if (point1.x == point2.x)
    {
        // 在同一列
        // 向左遍历
        // 以 p1 的中心点向左遍历获取点集合
        List<Point> p1LeftChanel = getLeftChanel(point1, 0, pieceWidth);
        // 以 p2 的中心点向左遍历获取点集合
        p2LeftChanel = getLeftChanel(point2, 0, pieceWidth);
        Map<Point, Point> leftLinkPoints = getYLinkPoints(p1LeftChanel,
            p2LeftChanel, pieceWidth);
        // 向右遍历，不得超过 Board 的边框（有方块的地方）
        // 以 p1 的中心点向右遍历获取点集合
        p1RightChanel = getRightChanel(point1, widthMax, pieceWidth);
        // 以 p2 的中心点向右遍历获取点集合
        List<Point> p2RightChanel = getRightChanel(point2, widthMax,
            pieceWidth);
        Map<Point, Point> rightLinkPoints = getYLinkPoints(p1RightChanel,
            p2RightChanel, pieceWidth);
        result.putAll(leftLinkPoints);
        result.putAll(rightLinkPoints);
    }
    // point2 位于 point1 的右上角
    if (isRightUp(point1, point2))
    {
        // 获取 point1 向上遍历，point2 向下遍历时横向可以连接的点
        Map<Point, Point> upDownLinkPoints = getXLinkPoints(p1UpChanel,
            p2DownChanel, pieceWidth);
        // 获取 point1 向右遍历，point2 向左遍历时纵向可以连接的点
        Map<Point, Point> rightLeftLinkPoints = getYLinkPoints(
            p1RightChanel, p2LeftChanel, pieceHeight);
        // 获取以 p1 为中心的向上通道
        p1UpChanel = getUpChanel(point1, 0, pieceHeight);
        // 获取以 p2 为中心的向上通道
        p2UpChanel = getUpChanel(point2, 0, pieceHeight);
        // 获取 point1 向上遍历，point2 向上遍历时横向可以连接的点
        Map<Point, Point> upUpLinkPoints = getXLinkPoints(p1UpChanel,
            p2UpChanel, pieceWidth);
        // 获取以 p1 为中心的向下通道
        p1DownChanel = getDownChanel(point1, heightMax, pieceHeight);
        // 获取以 p2 为中心的向下通道
        p2DownChanel = getDownChanel(point2, heightMax, pieceHeight);
        // 获取 point1 向下遍历，point2 向下遍历时横向可以连接的点
```

```
            Map<Point, Point> downDownLinkPoints = getXLinkPoints(p1DownChanel,
                p2DownChanel, pieceWidth);
            // 获取以 p1 为中心的向右通道
            p1RightChanel = getRightChanel(point1, widthMax, pieceWidth);
            // 获取以 p2 为中心的向右通道
            List<Point> p2RightChanel = getRightChanel(point2, widthMax,
                pieceWidth);
            // 获取 point1 向右遍历，point2 向右遍历时纵向可以连接的点
            Map<Point, Point> rightRightLinkPoints = getYLinkPoints(
                p1RightChanel, p2RightChanel, pieceHeight);
            // 获取以 p1 为中心的向左通道
            List<Point> p1LeftChanel = getLeftChanel(point1, 0, pieceWidth);
            // 获取以 p2 为中心的向左通道
            p2LeftChanel = getLeftChanel(point2, 0, pieceWidth);
            // 获取 point1 向左遍历，point2 向右遍历时纵向可以连接的点
            Map<Point, Point> leftLeftLinkPoints = getYLinkPoints(p1LeftChanel,
                p2LeftChanel, pieceHeight);
            result.putAll(upDownLinkPoints);
            result.putAll(rightLeftLinkPoints);
            result.putAll(upUpLinkPoints);
            result.putAll(downDownLinkPoints);
            result.putAll(rightRightLinkPoints);
            result.putAll(leftLeftLinkPoints);
        }
        // point2 位于 point1 的右下角
        if (isRightDown(point1, point2))
        {
            // 获取 point1 向下遍历，point2 向上遍历时横向可连接的点
            Map<Point, Point> downUpLinkPoints = getXLinkPoints(p1DownChanel,
                p2UpChanel, pieceWidth);
            // 获取 point1 向右遍历，point2 向左遍历时纵向可连接的点
            Map<Point, Point> rightLeftLinkPoints = getYLinkPoints(
                p1RightChanel, p2LeftChanel, pieceHeight);
            // 获取以 p1 为中心的向上通道
            p1UpChanel = getUpChanel(point1, 0, pieceHeight);
            // 获取以 p2 为中心的向上通道
            p2UpChanel = getUpChanel(point2, 0, pieceHeight);
            // 获取 point1 向上遍历，point2 向上遍历时横向可连接的点
            Map<Point, Point> upUpLinkPoints = getXLinkPoints(p1UpChanel,
                p2UpChanel, pieceWidth);
            // 获取以 p1 为中心的向下通道
            p1DownChanel = getDownChanel(point1, heightMax, pieceHeight);
            // 获取以 p2 为中心的向下通道
            p2DownChanel = getDownChanel(point2, heightMax, pieceHeight);
            // 获取 point1 向下遍历，point2 向下遍历时横向可连接的点
            Map<Point, Point> downDownLinkPoints = getXLinkPoints(p1DownChanel,
                p2DownChanel, pieceWidth);
            // 获取以 p1 为中心的向左通道
            List<Point> p1LeftChanel = getLeftChanel(point1, 0, pieceWidth);
            // 获取以 p2 为中心的向左通道
            p2LeftChanel = getLeftChanel(point2, 0, pieceWidth);
            // 获取 point1 向左遍历，point2 向左遍历时纵向可连接的点
            Map<Point, Point> leftLeftLinkPoints = getYLinkPoints(p1LeftChanel,
                p2LeftChanel, pieceHeight);
            // 获取以 p1 为中心的向右通道
            p1RightChanel = getRightChanel(point1, widthMax, pieceWidth);
            // 获取以 p2 为中心的向右通道
            List<Point> p2RightChanel = getRightChanel(point2, widthMax,
                pieceWidth);
            // 获取 point1 向右遍历，point2 向右遍历时纵向可以连接的点
            Map<Point, Point> rightRightLinkPoints = getYLinkPoints(
```

```
                p1RightChanel, p2RightChanel, pieceHeight);
            result.putAll(downUpLinkPoints);
            result.putAll(rightLeftLinkPoints);
            result.putAll(upUpLinkPoints);
            result.putAll(downDownLinkPoints);
            result.putAll(leftLeftLinkPoints);
            result.putAll(rightRightLinkPoints);
        }
        return result;
    }
```

上面的程序中粗体字代码分别调用了 getYLinkPoints、getXLinkPoints 方法来收集各种可能出现的连接路径，getYLinkPoints、getXLinkPoints 两种方法的代码如下。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 遍历两个集合，先判断第一个集合的元素的 X 坐标与另一个集合中的元素 X 坐标相同(纵向)，
 * 如果相同，即在同一列，再判断是否有障碍，没有则加到结果的 Map 中去
 * @param p1Chanel
 * @param p2Chanel
 * @param pieceHeight
 * @return
 */
private Map<Point, Point> getYLinkPoints(List<Point> p1Chanel,
    List<Point> p2Chanel, int pieceHeight)
{
    Map<Point, Point> result = new HashMap<Point, Point>();
    for (int i = 0; i < p1Chanel.size(); i++)
    {
        Point temp1 = p1Chanel.get(i);
        for (int j = 0; j < p2Chanel.size(); j++)
        {
            Point temp2 = p2Chanel.get(j);
            // 如果 X 坐标相同(在同一列)
            if (temp1.x == temp2.x)
            {
                // 没有障碍，放到 map 中去
                if (!isYBlock(temp1, temp2, pieceHeight))
                {
                    result.put(temp1, temp2);
                }
            }
        }
    }
    return result;
}
/**
 * 遍历两个集合，先判断第一个集合的元素的 Y 坐标与另一个集合中的元素 Y 坐标相同(横向)，
 * 如果相同，即在同一行，再判断是否有障碍，没有 则加到结果的 map 中去
 * @param p1Chanel
 * @param p2Chanel
 * @param pieceWidth
 * @return 存放可以横向直线连接的连接点的键值对
 */
private Map<Point, Point> getXLinkPoints(List<Point> p1Chanel,
    List<Point> p2Chanel, int pieceWidth)
{
    Map<Point, Point> result = new HashMap<Point, Point>();
    for (int i = 0; i < p1Chanel.size(); i++)
    {
        // 从第一通道中取一个点
```

```
            Point temp1 = p1Chanel.get(i);
            // 再遍历第二个通道，看下第二通道中是否有点可以与 temp1 横向相连
            for (int j = 0; j < p2Chanel.size(); j++)
            {
                Point temp2 = p2Chanel.get(j);
                // 如果 y 坐标相同(在同一行)，再判断它们之间是否有直接障碍
                if (temp1.y == temp2.y)
                {
                    if (!isXBlock(temp1, temp2, pieceWidth))
                    {
                        // 没有障碍则直接加到结果的 map 中
                        result.put(temp1, temp2);
                    }
                }
            }
        }
        return result;
    }
```

经过上面的处理之后，getLinkPoints(Point point1, Point point2,int pieceWidth, int pieceHeight)方法可以找出 point1、point2 两个点之间的所有可能的连接情况，该方法返回一个 Map 对象，Map 中每个 key-value 对代表一种连接情况，其中 key 代表第一个连接点，value 代表第二个连接点。

当 point1、point2 之间有多种连接情况时，程序还需要找出所有连接情况中的最短路径，link(Piece p1, Piece p2)方法中④号粗体字代码下调用了 getShortcut(p1Point, p2Point, turns,getDistance(p1Point, p2Point));方法进行处理，下面进行详细分析。

18.6.10 找出最短距离

为了找出所有连接情况中的最短路径，程序实现可分为两步。

① 遍历转折点 Map 中的所有 key-value 对，与原来选择的两个点构成一个 LinkInfo。每个 LinkInfo 代表一条完整的连接路径，并将这些 LinkInfo 收集成一个 List 集合

② 遍历第一步得到的 List<LinkInfo>集合，计算每个 LinkInfo 中连接全部连接点的总距离，选与最短距离相差最小的 LinkInfo 返回即可。

下面的方法实现了上面的思路。

程序清单：codes\Link\src\org\crazyit\link\board\impl\GameServiceImpl.java

```
/**
 * 获取 p1 和 p2 之间最短的连接信息
 * @param p1
 * @param p2
 * @param turns 放转折点的 map
 * @param shortDistance 两点之间的最短距离
 * @return p1 和 p2 之间最短的连接信息
 */
private LinkInfo getShortcut(Point p1, Point p2, Map<Point, Point> turns,
    int shortDistance)
{
    List<LinkInfo> infos = new ArrayList<LinkInfo>();
    // 遍历结果 Map,
    for (Point point1 : turns.keySet())
    {
        Point point2 = turns.get(point1);
        // 将转折点与选择点封装成 LinkInfo 对象，放到 List 集合中
```

```
                infos.add(new LinkInfo(p1, point1, point2, p2));
            }
            return getShortcut(infos, shortDistance);
        }
        /**
         * 从 infos 中获取连接线最短的那个 LinkInfo 对象
         * @param infos
         * @return 连接线最短的那个 LinkInfo 对象
         */
        private LinkInfo getShortcut(List<LinkInfo> infos, int shortDistance)
        {
            int temp1 = 0;
            LinkInfo result = null;
            for (int i = 0; i < infos.size(); i++)
            {
                LinkInfo info = infos.get(i);
                // 计算出几个点的总距离
                int distance = countAll(info.getLinkPoints());
                // 将循环第一个的差距用 temp1 保存
                if (i == 0)
                {
                    temp1 = distance - shortDistance;
                    result = info;
                }
                // 如果下一次循环的值比 temp1 的还小，则用当前的值作为 temp1
                if (distance - shortDistance < temp1)
                {
                    temp1 = distance - shortDistance;
                    result = info;
                }
            }
            return result;
        }
        /**
         * 计算 List<Point>中所有点的距离总和
         * @param points 需要计算的连接点
         * @return 所有点的距离的总和
         */
        private int countAll(List<Point> points)
        {
            int result = 0;
            for (int i = 0; i < points.size() - 1; i++)
            {
                // 获取第 i 个点
                Point point1 = points.get(i);
                // 获取第 i + 1 个点
                Point point2 = points.get(i + 1);
                // 计算第 i 个点与第 i + 1 个点的距离，并添加到总距离中
                result += getDistance(point1, point2);
            }
            return result;
        }
        /**
         * 获取两个 LinkPoint 之间的最短距离
         * @param p1 第一个点
         * @param p2 第二个点
         * @return 两个点的距离距离总和
         */
        private int getDistance(Point p1, Point p2)
        {
            int xDistance = Math.abs(p1.x - p2.x);
```

```
        int yDistance = Math.abs(p1.y - p2.y);
        return xDistance + yDistance;
    }
```

至此，连连看游戏中两个方块可能相连的所有情况都处理完成了，应用程序即可调用GameServiceImpl 所提供的 link(Piece p1, Piece p2)方法来判断两个方块是否可以相连了，这个过程也是编写该游戏最烦琐的地方。

通过对连连看游戏的分析与开发，读者应该发现编写一个游戏并没有想象的那么难，但也不像想象的那么简单。开发者需要冷静、条理化的思维，先分析游戏中所有可能出现的情况，然后在程序中对所有情况进行判断并进行相应的处理。

注意：

本程序的 GameService 组件的实现思路基本来自于《疯狂 Java 实战演义》第 7 章的实现思路，笔者无法保证这种实现方式为最优算法。这种算法实现起来有些烦琐，但它的条理十分清晰，非常适合初、中级程序员学习。



18.7 本章小结

本章开发了一个非常常见的单机休闲类游戏：Android 版的连连看，开发这个流行的小游戏难度适中，而且能充分激发学习热情，对于 Android 学习者来说是一个不错的选择。学习本章需要重点掌握单机游戏的界面分析与数据建模的能力：游戏玩家眼中看到的是游戏界面；但开发者眼中看到的应该是数据模型。除此之外，单机游戏通常总会有一个比较美观的界面，因此通常都需要通过自定义 View 来实现游戏主界面。连连看游戏中需要判断两个方块（图片）是否可以相连，这需要开发者对两个方块的位置分门别类地进行处理，并针对不同的情况提供相应的实现，这也是开发单机游戏需要重点掌握的能力。

CHAPTER

19

第19章 电子拍卖系统

本章要点

- Android 应用与传统应用整合的意义
- Android 应用整合传统应用的方式
- 基于 HttpClient、JSON 数据交换的整合方式
- JSON 基本知识
- JSON 语法
- 开发服务器端生成 JSON 响应的 Servlet
- 开发 Android 客户端界面
- 使用 HttpClient 发送请求
- 使用 HttpClient 获取服务器响应
- 将服务器响应转换为 JSON 对象或数组
- 通过 Android 客户端加载服务器响应

本章介绍了一个实用的 Android 应用：电子拍卖系统。本章所介绍的应用不再是普通的 Android 项目，而是一个 Android + Struts 2 + Spring 3+ Hibernate 4 整合的应用。Struts 2 + Spring 3 + Hibernate 4 提供了一个 B/S 结构的电子拍卖系统，对于使用 PC 的用户而言，他们可以使用浏览器来访问该系统；对于使用 Android 手机的用户而言，可以选择安装 Android 客户端程序，这样即可通过手机来使用该拍卖系统。

与一般图书上所介绍的 toy 项目不同，本应用的服务器端采用了完整的 Java EE 应用架构：技术实现上依赖于最流行的 Struts 2 + Spring + Hibernate 组合；应用架构采用了最流行的、具有高度可扩展性的控制器层 + 业务逻辑层 + DAO 层的分层架构。Android 客户端通过网络与服务器端的控制器组件交互，整个应用具有极好的示范性。

本应用的用户界面兼顾了手机和平板电脑两种设备，因此必须为手机屏幕和平板电脑屏幕设计用户界面，但由于该 Android 应用界面大量采用了 Fragment，而 Fragment 代表了可复用的“Activity 片段”，因此兼顾两种屏幕的界面复用了共同的 Fragment。



提示：

Android 系统发展的大势是与传统服务器应用系统整合，因为手机的硬件资源毕竟是有限的：计算能力有限，存储能力也有限。所以 Android 应用更适合作为应用的客户端：手机携带方便，可以随时随地开机运行应用，而且可以随时访问网络，并通过网络与服务器端应用交互、获取服务器端的数据。在未来的几年内，运行在手机上的电子商务客户端（实际上淘宝网已有了手机客户端）、证券系统客户端、金融系统客户端将会大量出现。

由于本章介绍的重点是 Android 开发，因此将会详细介绍 Android 应用的开发，以及如何让 Android 应用与远程服务器交互；本章不会全面、详细介绍服务器端的 Spring、Hibernate 开发，这些内容不是短短的一章可以讲完的。如果读者需要详细学习 Spring、Hibernate 相关知识，请参考疯狂 Java 体系的《轻量级 Java EE 企业应用实战》。

19.1　系统功能简介和架构设计

本章介绍的应用不再是简单的单机小应用，而是一个 Android 与传统服务器应用整合的应用，在这个应用中，服务器端程序依然保持了良好的应用架构，而客户端则使用 Android 程序充当。

➤➤19.1.1　系统功能简介

本章所介绍的系统是一个功能不太复杂的电子拍卖系统，本系统从实际电子商务平台上抽取，只取出其中部分核心功能实现，以作为示范应用，向读者展示一种良好的程序架构。

本章的电子拍卖系统其实就是一个电子商务平台，只要将该系统部署在互联网上，全球的客户都可以在该系统上发布想售出的商品，也可以对拍卖中的商品参与竞价。整个过程无须任何人工干预，由系统自动完成。

如果系统中提供与电子银行的接口，将可以通过电子银行的操作，实现买家对卖家的自动付款。一旦付款成功，就可以利用全球物流供应系统将拍卖物品发送到买家手中。可见，

这种电子拍卖系统是一种开放式的、成本极其低廉的系统，大部分工作无须人工干预，系统自动完成管理。当然，由于本系统是一个示范系统，因此不提供与电子银行的接口，只是模拟了用户添加拍卖物品，用户参与拍卖的基本行为。拍卖结束后，系统会判断物品是否被最高竞价者获得。

该电子拍卖系统模拟了淘宝系统部分功能，抽取了实际电子拍卖系统部分功能，但没有提供如个人身份认证、信用管理等细节问题，本系统主要实现了电子拍卖系统中的核心功能。

本项目的服务器端是一个完整的 Java EE 项目，服务器端采用了控制器层、业务逻辑层、DAO 层的分层架构。Android 客户端应用只负责与服务器的控制器组件交互，Android 应用采用 Apache HttpClient 向服务器端的控制器发送请求并获取服务器响应，这样即可实现 Android 系统与电子拍卖系统之间的通信。

本系统要求用户参与拍卖之前，必须登录系统。本系统提供了系统登录验证，登录验证在服务器端通过 Filter 实现，Filter 拦截用户请求，并判断 Session 中是否保存了当前用户 ID，如果保存了用户 ID，即该用户已经登录，否则没有登录。

对于物品的管理，本系统可以查询拍卖物品，添加拍卖物品，增加物品种类，竞价处理，以及发送邮件通知用户所参与的竞价。

- 注册用户可以添加用户物品，添加物品种类。添加之前必须登录系统。
- 注册用户可以浏览当前拍卖中的物品，以及流拍的物品。
- 注册用户可以参与竞价，参与的竞价系统将通过邮件通知用户。

19.1.2 系统架构设计

本系统的服务器采用 Java EE 的分层结构，分为视图层、控制器层、业务逻辑层和 DAO 层。分层体系将业务规则、数据访问等工作放到中间层处理，客户端不直接与数据库交互，而是通过控制器与中间层建立连接，再由中间层与数据库交互。

中间层采用 Struts 2+Spring 3+Hibernate 4，为了分离控制层与业务逻辑层，又可细分为：

- 控制器层，就是 MVC 模式里面的“C”（Controller），负责表现层与业务逻辑层的交互，调用业务逻辑层，并将业务数据返回给表现层来显示。MVC 框架采用流行的 Struts 2。
- Service 层（业务逻辑层），负责实现业务逻辑，对 DAO 对象进行正面模式的封装。
- DAO 层（数据访问对象层），负责与持久化对象交互，封装了数据的增、删、查、改原子操作。
- PO 层（持久化对象层），通过实体/关系映射工具将关系型数据库的数据映射成对象，实现以面向对象方式操作数据库，这个系统采用 Hibernate 作为 O/R Mapping 框架。

本系统使用 MySQL 数据库存放数据。

服务器端应用的总体架构如图 19.1 所示。

当采用 Android 应用作为客户端时，Android 应用可以通过网络与服务器端交互，Android 应用将会通过 Apache HttpClient 向服务器的控制器发送请求，并获取服务器响应，服务器响应将会采用 JSON 数据格式——可以更有效地进行数据交互。

Android 应用向服务器端的控制器发送请求，此处的控制器并不是 Struts 2 的 Action，而是直接采用 Servlet 充当。在 Servlet 3.0 规范中，开发 Servlet 已经不再需要在 web.xml 文件

中进行配置了，因此开发起来十分方便。

图 19.2 显示了 Android 客户端与服务器整合的架构。

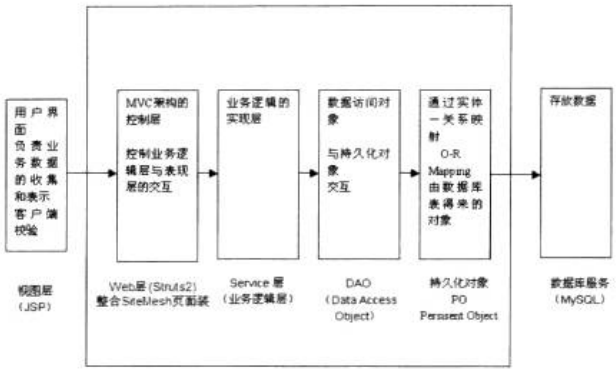


图 19.1　系统架构图

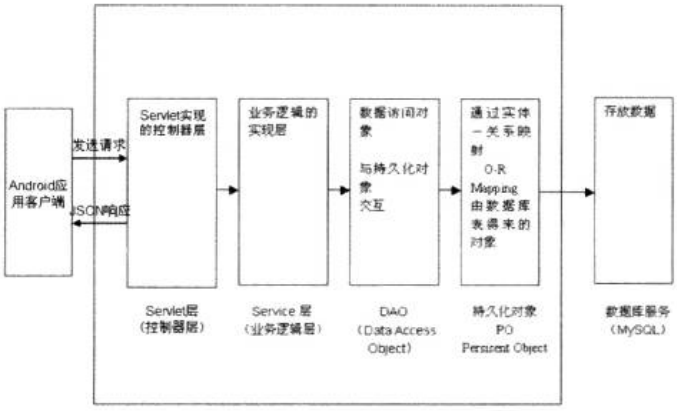


图 19.2　Android 应用与 Java EE 整合的架构

细心的读者可能已经发现：如图 19.1 与图 19.2 所示的结构十分相似，服务器端应用的结构基本不需要改变，应用只要在传统 Java EE 应用的基础上增加系列 Servlet，这些 Servlet 负责向 Android 客户端提供响应即可。

图 19.1 与图 19.2 十分相似正好说明了 Java EE 应用架构的优势：当整个应用的某一层需要改变或重构时，应用系统能最大限度地“复用”以前的应用组件，而不需要重新开发，这样才能保证以前编写的项目代码具有高度的保值性。也正是由于上面两个结构图的相似性，可让我们非常快速地为该应用增加各种平台的客户端系统。

19.2　JSON 简介

Android 与服务器端通信时需要一种合适的数据交换格式，本系统采用了 JSON 作为 Android 客户端与服务器的数据交换格式。



提示：

实际上还可以考虑使用 Web Service 来作为 Android 应用与服务器端的通信技术，对于 Web Service 而言，底层采用 XML 作为数据交换格式。

JSON 的全称是 JavaScript Object Notation，即 JavaScript 对象符号，它是一种轻量级的数据交换格式。JSON 的数据格式既适合人来读/写，也适合计算机本身解析和生成。最早的时候，JSON 是 JavaScript 语言的数据交换格式，后来慢慢发展成一种语言无关的数据交换格式，这一点非常类似于 XML。

JSON 主要在类似于 C 的编程语言中广泛使用，这些语言包括 C、C++、C#、Java、JavaScript、Perl、Python 等。JSON 提供了多种语言之间完成数据交换的能力，因此，JSON 也是一种非常理想的数据交换格式。JSON 主要有如下两种数据结构。

- 由 key-value 对组成的数据结构，这种数据结构在不同的语言中有不同的实现。例如，在 JavaScript 中是一个对象，在 Java 中是一种 Map 结构，在 C 语言中则是一个 struct。在其他语言中，可能有 record、dictionary、hash table 等。
- 有序集合。这种数据结构在不同语言中，可能有 list、vector、数组和序列等实现。

上面的两种数据结构在不同的语言中都有对应的实现。因此，这种简便的数据表示方式完全可以实现跨语言，因此可以作为程序设计语言中通用的数据交换格式。在 JavaScript 中主要有两种 JSON 的语法，一种用于创建对象，另一种用于创建数组。

19.2.1 使用 JSON 语法创建对象

JSON 语法创建对象是一种更简单的方式，使用 JSON 语法可避免书写函数，也可避免使用 new 关键字，而是直接获取一个 JavaScript 对象。对于早期的 JavaScript 版本，如果要使用 JavaScript 创建一个对象，通常情况下可能会这样写：

```
// 定义一个函数，作为构造器
function Person(name, sex)
{
    this.name = name;
    this.sex = sex;
}
// 创建一个 Person 实例
var p = new Person('yeeku', 'male');
// 输出 Person 实例
alert(p.name);
```

从 JavaScript 1.2 开始，创建对象有了一种更快捷的语法，语法如下：

```
var p = {"name": 'yeeku',
         "sex" : 'male'};
alert(p);
```

这种语法就是一种 JSON 语法。显然，使用 JSON 语法创建对象更加简捷、方便。图 19.3 显示了这种语法示意。

从图 19.3 可以看出，创建对象 object 时，总以{开始，以}结束，对象的每个属性名和属性值之间以英文冒号（:）隔开，多个属性定义之间以英文逗号（,）隔开。语法格式如下：

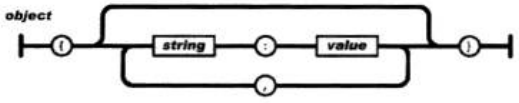


图 19.3 JSON 创建对象的语法示意

```
object =
{
    propertyName1 : propertyValue1 ,
```

```
        propertyName2 : propertyValue2 ,
        ...
}
```

必须注意的是，并不是每个属性定义后面都有英文逗号（,），必须在后面还有属性定义时才需要逗号（,）。因此，下面的对象定义是错误的：

```
person =
{
        name: 'yeeku',
        sex: 'male',
}
```

因为 sex 属性定义后多出一个英文逗号，最后一个属性定义的后面应直接以 } 结束，不再有英文的逗号（,）。

当然，使用 JSON 语法创建 JavaScript 对象时，属性值不仅可以是普通字符串，也可以是任何基本数据类型，还可以是函数、数组甚至是另外一个 JSON 语法创建的对象。例如：

```
person =
{
        name: 'yeeku',
        sex : 'male',
        //使用 JSON 语法为其指定一个属性
        son: {
                name:'nono',
                grade:1
        },
        //使用 JSON 语法为 person 直接分配一个方法
        info : function()
        {
                document.writeln("姓名：" + this.name + "性别：" + this.sex);
        }
}
```

➤➤19.2.2 使用 JSON 语法创建数组

使用 JSON 语法创建数组也是非常常见的情形，在早期的 JavaScript 语法里，我们通过如下方式来创建数组。

```
// 创建数组对象
var a = new Array();
// 为数组元素赋值
a[0] = 'yeeku';
// 为数组元素赋值
a[1] = 'nono';
```

或者，通过如下方式创建数组：

```
// 创建数组对象时直接赋值
var a = new Array('yeeku', 'nono');
```

但如果我们使用 JSON 语法，则可以通过如下方式创建数组：

```
// 使用 JSON 语法创建数组
var a = ['yeeku', 'nono'];
```

图 19.4 是 JSON 创建数组的语法示意。

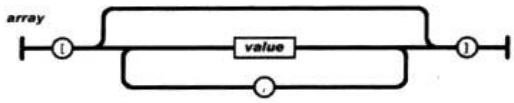


图 19.4 JSON 创建数组的语法示意

正如图 19.4 中所见到的，JSON 创建数组总是以英文方括号（[）开始，然后依次放入数组元素，元素与元素之间以英文逗号（,）隔开，最后一个数组元素后面不需要英文逗号，但以英文反方括号（]）结束。使用 JSON 创建数组的语法格式如下：

```
arr = [value1 , value 2 ...]
```

与 JSON 语法创建对象相似的是，数组的最后一个元素后面不能有逗号（,）。

鉴于 JSON 语法的简单易用，而且作为数据传输载体时，数据传输量更小，假设需要交换一个对象 person，其 name 属性为 yeeku，gender 属性为 male，age 属性为 29，使用 JSON 语法可以简化为如下形式：

```
person =
{
      name:'yeeku',
      gender:'male',
      age:29
}
```

但如果使用 XML 数据交换格式，则需要采用如下格式：

```
<person>
      <name>yeeku</name>
      <gender>male</gender>
      <age>29</age>
</person>
```

对比这两种表示方式，第一种方式明显比第二种方式更加简洁，数据传输量也更小。

19.2.3 Java 的 JSON 支持

当服务器返回一个满足 JSON 格式的字符串后，我们可以利用 JSON 项目提供的工具类将该字符串转化为 JSON 对象或 JSON 数组。

幸运的是，Android 系统内置了对 JSON 的支持，在 Android SDK 的 org.json 包下提供了 JSONArray、JSONObject、JSONStringer 和 JSONExeception 等类，通过这些类即可非常方便地完成 JSON 字符串与 JSONArray、JSONObject 之间的相互转换。

JDK 默认并未提供对 JSON 的支持，不过我们可以登录 http://www.json.org/java/index.html，在该页面可以看到 JSONArray、JSONObject、JSONStringer 和 JSONExeception 等类的源代码，读者可以自行下载这些源代码，把它们加入项目中即可使用。



提示： 笔者已经下载了这些源代码，并把它们编译成了*.class 文件，将它们打包成了 json.jar 包，该 JAR 包位于光盘的 codes\19\auction\WEB-INF\lib 目录下。读者直接把该 JAR 包添加到项目的类加载路径下即可使用。

Java 的 JSON 支持主要依赖于 JSONArray 和 JSONObject 两个类，其中：

- JSONArray 代表一个 JSON 数组，它可完成 Java 集合（集合元素可以是对象）与

JSON 字符串之间的相互转换。

- JSONObject 代表了一个 JSON 对象，它可完成 Java 对象与 JSON 字符串之间的相互转换。

19.3 发送请求的工具类

本系统采用 Apache HttpClient 与远程服务器通信，为了简化 HttpClient 的用法，本系统定义了一个工具类对 HttpClient 进行封装，该工具类定义了如下两个方法来发送请求。

- getRequest()：发送 GET 请求。
- postRequest()：发送 POST 请求。

该工具类的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\util\HttpUtil.java

```
public class HttpUtil
{
    // 创建 HttpClient 对象
    public static HttpClient httpClient = new DefaultHttpClient();
    public static final String BASE_URL =
        "http://192.168.1.88:8888/auction/android/";
    /**
     *
     * @param url 发送请求的 URL
     * @return 服务器响应字符串
     * @throws Exception
     */
    public static String getRequest(final String url)
        throws Exception
    {
        FutureTask<String> task = new FutureTask<String>(
        new Callable<String>()
        {
            @Override
            public String call() throws Exception
            {
                // 创建 HttpGet 对象。
                HttpGet get = new HttpGet(url);
                // 发送 GET 请求
                HttpResponse httpResponse = httpClient.execute(get);
                // 如果服务器成功地返回响应
                if (httpResponse.getStatusLine()
                        .getStatusCode() == 200)
                {
                    // 获取服务器响应字符串
                    String result = EntityUtils
                            .toString(httpResponse.getEntity());
                    return result;
                }
                return null;
            }
        });
        new Thread(task).start();
        return task.get();
    }
    /**
     * @param url 发送请求的 URL
```

```
     * @param params 请求参数
     * @return 服务器响应字符串
     * @throws Exception
     */
    public static String postRequest(final String url
            , final Map<String ,String> rawParams)throws Exception
    {
        FutureTask<String> task = new FutureTask<String>(
        new Callable<String>()
        {
            @Override
            public String call() throws Exception
            {
                // 创建 HttpPost 对象。
                HttpPost post = new HttpPost(url);
                // 如果传递参数个数比较多，可以对传递的参数进行封装
                List<NameValuePair> params =
                        new ArrayList<NameValuePair>();
                for(String key : rawParams.keySet())
                {
                    //封装请求参数
                    params.add(new BasicNameValuePair(key
                            , rawParams.get(key)));
                }
                // 设置请求参数
                post.setEntity(new UrlEncodedFormEntity(
                        params, "gbk"));
                // 发送 POST 请求
                HttpResponse httpResponse = httpClient.execute(post);
                // 如果服务器成功地返回响应
                if (httpResponse.getStatusLine()
                        .getStatusCode() == 200)
                {
                    // 获取服务器响应字符串
                    String result = EntityUtils
                            .toString(httpResponse.getEntity());
                    return result;
                }
                return null;
            }
        });
        new Thread(task).start();
        return task.get();
    }
}
```

上面的 HttpUtil 中两行粗体字代码定义了 getRuquest()和 postRequest()两个方法，这两个方法用于向服务器发送请求，方法返回服务器的响应。在 HttpUtil 中提供这两个方法之后，接下来在 Android 应用中只要调用这两个方法即可实现与服务器的通信。

19.4 用户登录

使用该电子拍卖系统之前，用户必须先登录系统，用户在 Android 应用中输入用户名、密码，单击“登录”按钮，程序即可通过 HttpUtil 向服务器发送请求，通过服务器来验证用户所输入的用户名、密码是否正确。

19.4.1 处理登录的 Servlet

处理用户登录的 Servlet 只是前端控制器，它的作用只有三个：

- 获取请求参数。
- 调用业务逻辑组件的方法来处理用户请求。
- 根据处理结果来生成输出。

由于本项目的服务器组件都是部署在 Spring 容器中的，因此程序需要获取 Spring 容器中的业务逻辑组件，而不是自行创建业务逻辑组件。

由于所有 Servlet 都需要获取 Spring 容器中的业务逻辑组件，因此本程序提供了一个 Servlet 基类，该基类中定义了一个方法来获取 Web 应用中的 Spring 容器。该 Servlet 基类的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\base\BaseServlet.java

```
public class BaseServlet extends HttpServlet
{
    private ApplicationContext ctx;
    public void init(ServletConfig config)
        throws ServletException
    {
        super.init(config);
        // 获取 Web 应用中的 ApplicationContext 实例
        ctx = WebApplicationContextUtils
            .getWebApplicationContext(getServletContext());
    }
    // 返回 Web 应用中的 Spring 容器
    public ApplicationContext getCtx()
    {
        return this.ctx;
    }
}
```

正如上面的粗体字代码所看到的，该 Servlet 基类的粗体字代码定义了一个 getCtx()方法，通过该方法即可获取 Spring 容器，这意味着应用中的所有 Servlet 都可通过该方法来获取 Spring 容器。接下来服务器端的所有应用程序只要继承该 Servlet，并通过该方法来获取 Spring 容器即可。

下面是处理用户登录的 Servlet 类代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\LoginServlet.java

```
@WebServlet(urlPatterns="/android/login.jsp")
public class LoginServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
        HttpServletResponse response)
        throws IOException , ServletException
    {
        String user = request.getParameter("user");
        String pass = request.getParameter("pass");
        // 获取系统的业务逻辑组件
        AuctionManager auctionManager = (AuctionManager)getCtx()
            .getBean("mgr");
        // 验证用户登录
        int userId = auctionManager.validLogin(user , pass);
        response.setContentType("text/html; charset=GBK");
```

```
            // 登录成功
            if (userId > 0)
            {
                request.getSession(true).setAttribute("userId" , userId);
            }
            try
            {
                // 把验证的 userId 封装成 JSONObject
                JSONObject jsonObj = new JSONObject()
                    .put("userId" , userId);
                // 输出响应
                response.getWriter().println(jsonObj.toString());
            }
            catch (JSONException ex)
            {
                ex.printStackTrace();
            }
        }
    }
```

该 Servlet 中第一行粗体字代码使用了@WebServlet 指定该 Servlet 的 URL 为/android/login.jsp。第二行粗体字代码调用了业务逻辑组件的 validateLogin()方法来处理用户登录。如果用户登录成功，程序将用户的 ID 放入 HTTP session 中，方便程序跟踪用户的登录状态。



提示： 如果读者对 Spring、Hibernate 等框架不熟悉也不要紧，读者可以直接使用 LoginServlet 访问系统数据库，并根据用户输入的用户名、密码返回该用户的 ID，该 Android 客户端一样可以正常工作。只是采用这种方式开发的应用不太正规，后期的扩展性很差而已。

19.4.2 用户登录

Android 客户端中用户登录界面有两个文本框，用于接收用户输入的用户名、密码信息，下面是用户登录的界面布局 XML 文档。

程序清单：codes\19\AuctionClient\res\layout\login.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="300dp"
    android:layout_height="match_parent"
    android:layout_gravity="center_horizontal"
    android:stretchColumns="1">
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitCenter"
    android:src="@drawable/logo"
    android:contentDescription="@string/hello"/>
<TextView
    android:text="@string/welcome"
    android:id="@+id/TextView"
    android:textSize="@dimen/label_font_size"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:padding="@dimen/title_padding"/>
```

```
<!-- 输入用户名的行 -->
<TableRow>
<TextView
      android:text="@string/user_name"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<EditText
      android:id="@+id/userEditText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:inputType="text"/>
</TableRow>
<!-- 输入密码的行 -->
<TableRow>
<TextView
      android:text="@string/user_pass"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<EditText
      android:text=""
      android:id="@+id/pwdEditText"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:inputType="textPassword"/>
</TableRow>
<!-- 定义登录、取消按钮的行 -->
<LinearLayout android:orientation="horizontal"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:gravity="center">
<Button
      android:id="@+id/bnLogin"
      android:text="@string/login"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<Button
      android:id="@+id/bnCancel"
      android:text="@string/cancel"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
</LinearLayout>
</TableLayout>
```

该界面布局的运行效果如图 19.5 所示。



图 19.5　用户登录界面

在图 19.5 所示的界面中输入用户名、密码之后，如果用户单击“登录”按钮将会激发登录处理，也就是通过 HttpUtil 向服务器发送请求。用户登录的 Activity 代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\Login.java

```
public class Login extends Activity
{
      // 定义界面中两个文本框
      EditText etName, etPass;
      // 定义界面中两个按钮
      Button bnLogin, bnCancel;
      @Override
      public void onCreate(Bundle savedInstanceState)
      {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);
        // 获取界面中两个编辑框
        etName = (EditText) findViewById(R.id.userEditText);
        etPass = (EditText) findViewById(R.id.pwdEditText);
        // 获取界面中的两个按钮
        bnLogin = (Button) findViewById(R.id.bnLogin);
        bnCancel = (Button) findViewById(R.id.bnCancel);
        // 为 bnCancal 按钮的单击事件绑定事件监听器
        bnCancel.setOnClickListener(new HomeListener(this));
        bnLogin.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 执行输入校验
                if (validate()) //①
                {
                    // 如果登录成功
                    if (loginPro()) //②
                    {
                        // 启动 Main Activity
                        Intent intent = new Intent(Login.this
                            , AuctionClientActivity.class);
                        startActivity(intent);
                        // 结束该 Activity
                        finish();
                    }
                    else
                    {
                        DialogUtil.showDialog(Login.this
                            , "用户名称或者密码错误，请重新输入！", false);
                    }
                }
            }
        });
    }
    private boolean loginPro()
    {
        // 获取用户输入的用户名、密码
        String username = etName.getText().toString();
        String pwd = etPass.getText().toString();
        JSONObject jsonObj;
        try
        {
            jsonObj = query(username, pwd);
            // 如果 userId 大于 0
            if (jsonObj.getInt("userId") > 0)
            {
                return true;
            }
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(this
                , "服务器响应异常，请稍后再试！", false);
            e.printStackTrace();
        }
        return false;
    }
    // 对用户输入的用户名、密码进行校验
```

```
    private boolean validate()
    {
        String username = etName.getText().toString().trim();
        if (username.equals(""))
        {
            DialogUtil.showDialog(this, "用户账户是必填项！", false);
            return false;
        }
        String pwd = etPass.getText().toString().trim();
        if (pwd.equals(""))
        {
            DialogUtil.showDialog(this, "用户口令是必填项！", false);
            return false;
        }
        return true;
    }
    // 定义发送请求的方法
    private JSONObject query(String username, String password)
        throws Exception
    {
        // 使用 Map 封装请求参数
        Map<String, String> map = new HashMap<String, String>();
        map.put("user", username);
        map.put("pass", password);
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "login.jsp";
        // 发送请求
        return new JSONObject(HttpUtil.postRequest(url, map));
    }
}
```

上面 Activity 的 query()方法中两行粗体字代码用于向指定 URL 发送请求，并将服务器响应封装成 JSONObject。

上面的程序为登录按钮的单击事件绑定了事件监听器，当用户单击“登录”按钮时，程序先执行输入校验，如上面的①号代码所示；接下来再执行登录处理，如上面的②号代码所示——程序调用了 loginPro 来处理用户的登录请求。

如果用户输入的用户名、密码不正确，系统将会调用 DialogUtil 来显示对话框，对话框提示登录失败。由于本系统经常需要显示各种对话框，因此程序专门把它定义成一个独立的类。DialogUtil 类的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\util\DialogUtil.java

```
public class DialogUtil
{
    // 定义一个显示消息的对话框
    public static void showDialog(final Context ctx
        , String msg , boolean goHome)
    {
        // 创建一个 AlertDialog.Builder 对象
        AlertDialog.Builder builder = new AlertDialog.Builder(ctx)
            .setMessage(msg).setCancelable(false);
        if(goHome)
        {
            builder.setPositiveButton("确定", new OnClickListener()
            {
                @Override
                public void onClick(DialogInterface dialog, int which)
                {
                    Intent i = new Intent(ctx , AuctionClientActivity.class);
```

```
                        i.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                        ctx.startActivity(i);
                    }
                });
            }
            else
            {
                builder.setPositiveButton("确定", null);
            }
            builder.create().show();
        }
        // 定义一个显示指定组件的对话框
        public static void showDialog(Context ctx , View view)
        {
            new AlertDialog.Builder(ctx)
                .setView(view).setCancelable(false)
                .setPositiveButton("确定", null)
                .create()
                .show();
        }
    }
```

如果登录成功，系统启动 AuctionClientActivity，这个 Activity 相当于系统主界面，程序用户可通过该界面提供的 ListView 进入各功能。

AuctionClientActivity 的界面布局文件可分为两个：为普通手机屏幕提供的应用界面和为平板电脑屏幕提供的界面。下面先看为普通手机屏幕提供的界面布局文件。

程序清单：codes\19\AuctionClient\res\layout\activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- 添加一个 Fragment -->
    <fragment android:name="org.crazyit.auction.client.AuctionListFragment"
        android:id="@+id/auction_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</LinearLayout>
```

上面程序界面的主要组件就是 AuctionListFragment，该 Fragment 显示一个 ListView，该 ListView 中每个列表项代表一个系统功能。

下面再看为平板电脑屏幕提供的界面布局文件。

程序清单：codes\19\AuctionClient\res\layout-sw600dp\activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- 定义一个水平排列的 LinearLayout，并指定使用中等分隔条 -->
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp"
    android:divider="?android:attr/dividerHorizontal"
    android:showDividers="middle">
    <!-- 添加一个 Fragment -->
```

```
        <fragment android:name="org.crazyit.auction.client.AuctionListFragment"
            android:id="@+id/auction_list"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1" />
        <!-- 添加一个 FrameLayout 容器 -->
        <FrameLayout
            android:id="@+id/auction_detail_container"
            android:layout_width="0dp"
            android:paddingLeft="10dp"
            android:layout_height="match_parent"
            android:layout_weight="3" />
    </LinearLayout>
```

上面的程序界面中同样包含一个 AuctionListFragment，除此之外，该程序界面中还包含一个 FrameLayout 容器，该容器负责装载对应功能的 Fragment 组件。

由于 layout 和 layout-sw600dp 目录下分别包含了 activity_main.xml 界面布局文件，因此 AuctionClientActivity 会根据屏幕尺寸自动加载不同目录下的界面布局文件。除此之外，该 Activity 必须根据界面布局来进行处理：如果程序加载 layout 目录下的 activity_main.xml 布局文件，用户点击功能项时，系统将会启动相应的 Activity 来显示功能；如果程序加载 layout-sw600dp 目录下的 activity_main.xml 布局文件，用户点击功能项时，系统将会使用 FrameLayout 加载相应功能的 Fragment。

AuctionClientActivity 的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\AuctionClientActivity.java

```
public class AuctionClientActivity extends Activity
    implements Callbacks
{
    // 定义一个旗标，用于标识该应用是否支持大屏幕
    private boolean mTwoPane;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        // 指定加载 R.layout.activity_main 对应的界面布局文件
        // 但实际上该应用会根据屏幕分辨率加载不同的界面布局文件
        setContentView(R.layout.activity_main);
        // 如果加载的界面布局文件中包含 ID 为 book_detail_container 的组件
        if (findViewById(R.id.auction_detail_container) != null)
        {
            mTwoPane = true;
            ((AuctionListFragment) getFragmentManager()
                .findFragmentById(R.id.auction_list))
                .setActivateOnItemClick(true);
        }
    }
    @Override
    public void onItemSelected(Integer id , Bundle bundle)
    {
        if (mTwoPane)
        {
            Fragment fragment = null;
            switch ((int) id)
            {
                // 查看竞得物品
                case 0:
                    // 创建 ViewItemFragment
```

```
                fragment = new ViewItemFragment();
                // 创建 Bundle，准备向 Fragment 传入参数
                Bundle arguments = new Bundle();
                arguments.putString("action", "viewSucc.jsp");
                // 向 Fragment 传入参数
                fragment.setArguments(arguments);
                break;
            // 浏览流拍物品
            case 1:
                // 创建 ViewItemFragment
                fragment = new ViewItemFragment();
                // 创建 Bundle，准备向 Fragment 传入参数
                Bundle arguments2 = new Bundle();
                arguments2.putString("action", "viewFail.jsp");
                // 向 Fragment 传入参数
                fragment.setArguments(arguments2);
                break;
            // 管理物品种类
            case 2:
                // 创建 ManageKindFragment
                fragment = new ManageKindFragment();
                break;
            // 管理物品
            case 3:
                // 创建 ManageItemFragment
                fragment = new ManageItemFragment();
                break;
            // 浏览拍卖物品（选择物品种类）
            case 4:
                // 创建 ChooseKindFragment
                fragment = new ChooseKindFragment();
                break;
            // 查看自己的竞标
            case 5:
                // 创建 ViewBidFragment
                fragment = new ViewBidFragment();
                break;
            case ManageItemFragment.ADD_ITEM:
                fragment = new AddItemFragment();
                break;
            case ManageKindFragment.ADD_KIND:
                fragment = new AddKindFragment();
                break;
            case ChooseKindFragment.CHOOSE_ITEM:
                fragment = new ChooseItemFragment();
                Bundle args = new Bundle();
                args.putLong("kindId", bundle.getLong("kindId"));
                fragment.setArguments(args);
                break;
            case ChooseItemFragment.ADD_BID:
                fragment = new AddBidFragment();
                Bundle args2 = new Bundle();
                args2.putInt("itemId", bundle.getInt("itemId"));
                fragment.setArguments(args2);
                break;
        }
        // 使用 fragment 替换 auction_detail_container 容器当前显示的 Fragment
        getFragmentManager().beginTransaction()
            .replace(R.id.auction_detail_container, fragment)
            .addToBackStack(null).commit();
    }
```

```
            else
            {
                Intent intent = null;
                switch ((int) id)
                {
                    // 查看竞得物品
                    case 0:
                        // 启动 ViewItem Activity
                        intent = new Intent(this, ViewItem.class);
                        // action 属性为请求的 Servlet 地址。
                        intent.putExtra("action", "viewSucc.jsp");
                        startActivity(intent);
                        break;
                    // 浏览流拍物品
                    case 1:
                        // 启动 ViewItem Activity
                        intent = new Intent(this, ViewItem.class);
                        // action 属性为请求的 Servlet 的 URL。
                        intent.putExtra("action", "viewFail.jsp");
                        startActivity(intent);
                        break;
                    // 管理物品种类
                    case 2:
                        // 启动 ManageKind Activity
                        intent = new Intent(this, ManageKind.class);
                        startActivity(intent);
                        break;
                    // 管理物品
                    case 3:
                        // 启动 ManageItem Activity
                        intent = new Intent(this, ManageItem.class);
                        startActivity(intent);
                        break;
                    // 浏览拍卖物品（选择物品种类）
                    case 4:
                        // 启动 ChooseKind Activity
                        intent = new Intent(this, ChooseKind.class);
                        startActivity(intent);
                        break;
                    // 查看自己的竞标
                    case 5:
                        // 启动 ViewBid Activity
                        intent = new Intent(this, ViewBid.class);
                        startActivity(intent);
                        break;
                }
            }
        }
    }
```

从上面的代码可以看出，如果在手机屏幕上运行，AuctionClientActivity 主要提供一个 ListView，当用户单击不同的列表项时，程序将会启动不同的 Activity，AuctionClientActivity 的运行效果如图 19.6 所示。

用户单击图 19.6 所示的“主菜单”（其实是 ListView）的任一列表项，系统将会启动对应的 Activity，从而允许用户执行相应的操作。

但如果在平板电脑上运行该实例，AuctionClientActivity 则提供一个 ListView 和一个 FrameLayout 容器，当用户单击不同的列表项时，程序将使用 FrameLayout 装载相应功能的 Fragment。在平板电脑上运行 AuctionClientActivity 的效果如图 19.7 所示。

用户单击图 19.6 所示的“主菜单”（其实是 ListView）的任一列表项，系统将会使用右边的 FrameLayout 装载相应功能的 Fragment。



图 19.6 系统主菜单



图 19.7 系统主界面

19.5 查看流拍物品

当用户单击查看“浏览流拍物品”时，程序将使用 FrameLayout 装载相应的 Fragment，启动相应的 Activity 装载相应的 Fragment，并显示系统中的流拍物品。

➤➤19.5.1 查看流拍物品的 Servlet

查看流拍物品的 Servlet 也只是前端控制器，它会调用 Spring 容器中业务逻辑组件的方法，通过该方法获取系统中的流拍物品。

该 Servlet 类的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewFailServlet.java

```
@WebServlet(urlPatterns="/android/viewFail.jsp")
public class ViewFailServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
        {
            // 获取业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 获取系统内所有流拍的物品
            List<ItemBean> items = auctionManager.getFailItems();
            JSONArray jsonArr= new JSONArray(items);
            response.setContentType("text/html; charset=GBK");
            response.getWriter().println(jsonArr.toString()); //①
        }
}
```

上面的程序中粗体字代码调用了 Spring 容器中的业务逻辑组件来获取系统中的流拍物品，程序的第二行粗体字代码把 Java 集合包装成 JSONArray 对象，程序中①号粗体字代码用于把 JSONArray 对象转换为 JSON 字符串，并作为响应输出到客户端。

如果直接向服务器的/android/viewFail.jsp 发送请求，将可以看到如图 19.8 所示的输出。

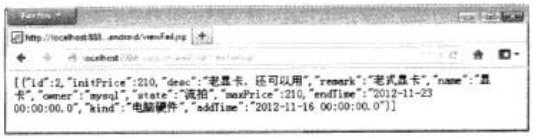


图 19.8　浏览流拍物品的 JSON 响应

图 19.8 显示的字符串就是典型的 JSON 格式字符串，Android 客户端程序只要调用 JSONArray 类的构造器即可把它转换为 JSONArray——就是一个 JSON 数组。



提示：

与前面登录的 Servlet 类似，如果读者对 Spring、Hibernate 等技术不熟，也可以直接在 Servlet 中访问数据库，并获取系统中所有流拍物品，只要该 Servlet 能获取到所有的流拍物品，并把它们转换为如图 19.8 所示的 JSON 字符串输出，该 Android 客户端就可正常工作。

19.5.2　查看流拍物品

下面是查看流拍物品的程序界面。

程序清单：codes\19\AuctionClient\res\layout\view_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:gravity="center"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<LinearLayout
      android:orientation="horizontal"
      android:gravity="center"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="@dimen/sub_title_margin">
<TextView
      android:id="@+id/view_titile"
      android:text="@string/view_succ"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<!-- 定义返回按钮 -->
<Button
      android:id="@+id/bn_home"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="@dimen/label_font_size"
      android:background="@drawable/home"/>
</LinearLayout>
<!-- 查看物品列表的 ListView -->
<ListView
      android:id="@+id/succList"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>
</LinearLayout>
```

上面的界面布局中的主要组件就是一个 ListView，用于显示多个物品。查看流拍物品的

Fragment 代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ViewItemFragment.java

```
public class ViewItemFragment extends Fragment
{
    Button bnHome;
    ListView succList;
    TextView viewTitle;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.view_item
            , container , false);
        // 获取界面上的“返回”按钮
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        succList = (ListView) rootView.findViewById(R.id.succList);
        viewTitle = (TextView) rootView.findViewById(R.id.view_titile);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        String action = getArguments().getString("action");
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + action;
        // 如果是查看流拍物品，修改标题
        if (action.equals("viewFail.jsp"))
        {
            viewTitle.setText(R.string.view_fail);
        }
        try
        {
            // 向指定 URL 发送请求，并把服务器响应转换成 JSONArray 对象
            JSONArray jsonArray = new JSONArray(HttpUtil
                .getRequest(url)); //①
            // 将 JSONArray 包装成 Adapter
            JSONArrayAdapter adapter = new JSONArrayAdapter(getActivity()
                , jsonArray, "name", true); //②
            succList.setAdapter(adapter);
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity(), "服务器响应异常，请稍后再试！",
            false);
            e.printStackTrace();
        }
        succList.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                int position, long id)
            {
                // 查看指定物品的详细情况
                viewItemDetail(position); //③
            }
        });
        return rootView;
    }
    private void viewItemDetail(int position)
    {
        // 加载 detail.xml 界面布局代表的视图
        View detailView = getActivity().getLayoutInflater()
            .inflate(R.layout.detail, null);
```

```
            // 获取 detail.xml 界面布局中的文本框
            TextView itemName = (TextView) detailView
                  .findViewById(R.id.itemName);
            TextView itemKind = (TextView) detailView
                  .findViewById(R.id.itemKind);
            TextView maxPrice = (TextView) detailView
                  .findViewById(R.id.maxPrice);
            TextView itemRemark = (TextView) detailView
                  .findViewById(R.id.itemRemark);
            // 获取被单击的列表项
            JSONObject jsonObj = (JSONObject) succList.getAdapter().getItem(
                  position);
            try
            {
                  // 通过文本框显示物品详情
                  itemName.setText(jsonObj.getString("name"));
                  itemKind.setText(jsonObj.getString("kind"));
                  maxPrice.setText(jsonObj.getString("maxPrice"));
                  itemRemark.setText(jsonObj.getString("desc"));
            }
            catch (JSONException e)
            {
                  e.printStackTrace();
            }
            DialogUtil.showDialog(getActivity(), detailView);
      }
}
```

上面的程序中①号粗体字代码用于向指定 URL 发送请求，并把服务器响应包装成 JSONArray 对象，这就完成了 Android 客户端与服务器的交互。JSONArray 对象的本质就是一个数组，它提供了如下常用方法。

- length()：返回该 JSON 数组的长度。
- optJSONObject(int index)：获取指定索引处的 JSONObject 对象。
- optJSONArray(int index)：获取指定所索引处的 JSONArray 对象。
- optXxx(int index)：获取指定索引处的数组元素。

对于开发者而言，当程序中拿到一个 JSONArray 对象之后，心中完全可以把它当成数组处理。

本程序提供了一个 JSONArrayAdapter 类，它的本质是一个 Adapter，该 Adapter 用于对 JSON 数组进行包装，并作为 ListView 的内容 Adapter。JSONArrayAdapter 类的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\JSONArrayAdapter.java

```
public class JSONArrayAdapter extends BaseAdapter
{
      private Context ctx;
      // 定义需要包装的 JSONArray 对象
      private JSONArray jsonArray;
      // 定义列表项显示 JSONObject 对象的哪个属性
      private String property;
      private boolean hasIcon;
      public JSONArrayAdapter(Context ctx
            , JSONArray jsonArray, String property
            , boolean hasIcon)
      {
            this.ctx = ctx;
            this.jsonArray = jsonArray;
            this.property = property;
```

```
        this.hasIcon = hasIcon;
    }
    @Override
    public int getCount()
    {
        return jsonArray.length();
    }
    @Override
    public Object getItem(int position)
    {
        return jsonArray.optJSONObject(position);
    }
    @Override
    public long getItemId(int position)
    {
        try
        {
            // 返回物品的 ID
            return ((JSONObject)getItem(position)).getInt("id");
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        return 0;
    }
    @Override
    public View getView(int position, View convertView, ViewGroup parent)
    {
        // 定义一个线性布局管理器
        LinearLayout linear = new LinearLayout(ctx);
        // 设置为水平的线性布局管理器
        linear.setOrientation(0);
        // 创建一个 ImageView
        ImageView iv = new ImageView(ctx);
        iv.setPadding(10, 0, 20, 0);
        iv.setImageResource(R.drawable.item);
        // 将图片添加到 LinearLayout 中
        linear.addView(iv);
        // 创建一个 TextView
        TextView tv = new TextView(ctx);
        try
        {
            // 获取 JSONArray 数组元素的 property 属性
            String itemName = ((JSONObject)getItem(position))
                .getString(property);
            // 设置 TextView 所显示的内容
            tv.setText(itemName);
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        tv.setTextSize(20);
        if (hasIcon)
        {
            // 将 TextView 添加到 LinearLayout 中
            linear.addView(tv);
            return linear;
        }
        else
```

```
            {
                tv.setTextColor(Color.BLACK);
                return tv;
            }
        }
    }
```

上面的 Adapter 负责为 ListView 提供列表项，ListView 的列表项可以是如下两种情况。

- 当 ListView 的列表项有图标时，该 Adapter 所提供的每个列表项是一个 LinearLayout（里面包括一个图标和一个 TextView）。
- 当 ListView 的列表项没有图标时，该 Adapter 提供的每个列表项就是一个普通的 TextView。

上面的 JSONArrayAdapter 可以多次复用，因此程序把它单独定义成一个 Adapter 类，这样可以多次使用该 Adapter 来封装 JSONArray 对象。

在平板电脑上查看系统流拍物品，将看到如图 19.9 所示的界面。

如图 19.9 所示列表只能看到流拍物品的物品名，如果用户希望看到该物品的详情，可以单击该物品对应的列表项，此时程序将会触发 ViewItemFragment 中的 viewItemDetail(position); 方法，如 ViewItem 类中的③号粗体字代码所示。

viewItemDetail(position)方法会使用对话框加载系统中的 detail.xml 界面布局文件，并显示该流拍物品的详情，当用户单击图 19.9 所示的列表项时，系统将会显示如图 19.10 所示的对话框。

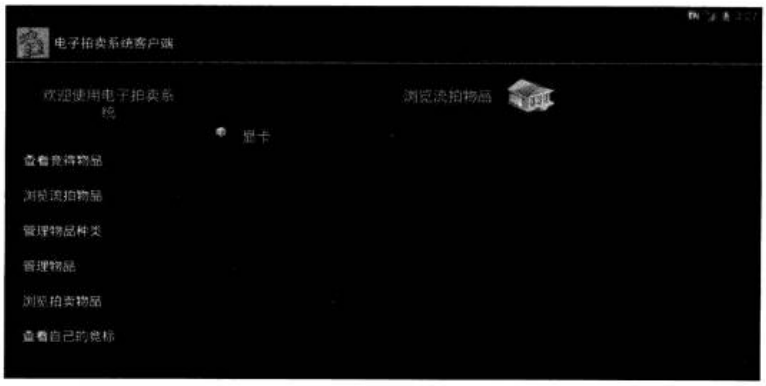


图 19.9　平板上查看流拍物品

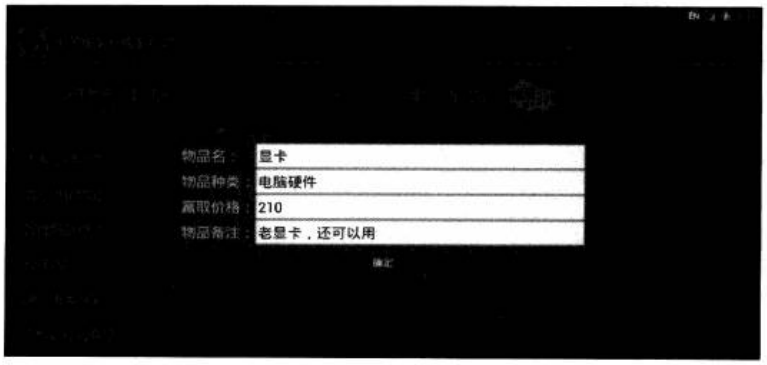


图 19.10　查看物品详情

上面的 ViewItemFragment 除了用于显示流拍物品之外，也可用于查看竞得物品，因为两

个功能的前端实现基本是一致的，只是 Android 需要调用的服务器端方法不同而已。

如果在手机屏幕上使用，系统还需要使用一个 Activity 来“盛装”该 Fragment，由于该系统中有大量 Fragment 需要使用 Activity 来显示，因此本系统专门开发了一个 FragmentActivity，它只是用于显示一个 Fragment，该 Activity 的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\app\base\FragmentActivity.java

```
public abstract class FragmentActivity extends Activity
{
    private static final int ROOT_CONTAINER_ID = 0x90001;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        LinearLayout layout = new LinearLayout(this);
        setContentView(layout);
        layout.setId(ROOT_CONTAINER_ID);
        getFragmentManager().beginTransaction()
            .replace(ROOT_CONTAINER_ID , getFragment())
            .commit();
    }
    protected abstract Fragment getFragment();
}
```

上面的粗体字代码定义了一个抽象方法，该抽象方法将会返回一个 Fragment，而该 Activity 的作用就是加载、显示该 Fragment。

FragmentActivity 用于被其他 Activity 继承，继承它的 Activity 只要重写 getFragment()方法即可，下面是 ViewItem Activity 的代码。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ViewItem.java

```
public class ViewItem extends FragmentActivity
{
    // 重写 getFragment()方法，该 Activity 显示该方法返回的 Fragment
    @Override
    protected Fragment getFragment()
    {
        ViewItemFragment fragment = new ViewItemFragment();
        Bundle arguments = new Bundle();
        arguments.putString("action"
            , getIntent().getStringExtra("action"));
        fragment.setArguments(arguments);
        return fragment;
    }
}
```

从上面的代码不难看出，ViewItem Activity 只是显示 ViewItemFragment 而已——显示该 Fragment 之前，向该 Fragment 传入参数。

在普通手机上查看流拍物品，将可以看到如图 19.11 所示界面。



图 19.11 手机上查看流拍物品

19.6 管理物品种类

管理物品种类包括浏览系统中的物品种类，添加物品种类两大主要功能。Android 客户端主要充当用户交互的客户端：显示系统中物品种类；添加种类时提供输入框供用户输入种类名称、种类描述等必要信息。

19.6.1 浏览物品种类的 Servlet

浏览物品种类的 Servlet 同样只是充当控制器，调用业务逻辑组件的业务方法来获取系统中的物品种类，该 Servlet 的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewKindServlet.java

```
@WebServlet(urlPatterns="/android/viewKind.jsp")
public class ViewKindServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
        {
            // 获取业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 获取系统中所有物品种类
            List<KindBean> kinds = auctionManager.getAllKind();
            // 将所有物品种类包装成 JSONArray
            JSONArray jsonArr= new JSONArray(kinds);
            response.setContentType("text/html; charset=GBK");
            // 将 JSONArray 转换成 JSON 字符串后输出到客户端
            response.getWriter().println(jsonArr.toString());
        }
}
```

与前一个 Servlet 的实现基本相似，该 Servlet 也是调用 Spring 容器业务逻辑组件相应的业务方法，并把方法返回的字符串包装成 JSONArray，再把 JSONArray 转换为字符串作为服务器响应。

直接使用浏览器浏览上面的 Servlet，将可以看到如图 19.12 所示的 JSON 字符串输出。

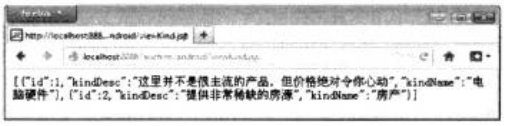


图 19.12　JSON 字符串响应

19.6.2 查看物品种类

在 Android 客户端查看物品种类，只要先通过 HttpUtil 向服务器发送请求，并把服务器响应字符串转换成 JSONArray 对象，在使用 Adapter 包装 JSONArray 对象，并使用 ListView 进行显示即可。

查看物品种类的界面布局代码如下。

程序清单：codes\19\AuctionClient\res\layout\manage_kind.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:gravity="center"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
<LinearLayout
    android:orientation="horizontal"
    android:gravity="center"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/sub_title_margin">
    <LinearLayout
        android:orientation="vertical"
        android:gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    <TextView
        android:text="@string/manage_kind"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <!-- 添加种类的按钮 -->
    <Button
        android:id="@+id/bnAdd"
        android:layout_width="85dp"
        android:layout_height="30dp"
        android:background="@drawable/add_kind"/>
    </LinearLayout>
    <Button
        android:id="@+id/bn_home"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="@dimen/label_font_size"
        android:background="@drawable/home"/>
    </LinearLayout>
    <!-- 显示种类列表的 ListView -->
    <ListView
        android:id="@+id/kindList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
    </LinearLayout>
```

上面的界面布局中定义了一个 ListView 来显示系统中的物品种类，程序只要把服务器返回的物品种类包装成 Adapter、并使用该 ListView 来显示所有物品种类即可。下面是显示物品种类的 Fragment 代码。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ManageKindFragment.java

```
public class ManageKindFragment extends Fragment
{
    public static final int ADD_KIND = 0x1007;
    Button bnHome , bnAdd;
    ListView kindList;
    Callbacks mCallbacks;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.manage_kind
            , container , false);
        // 获取界面布局上的两个按钮
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        bnAdd = (Button) rootView.findViewById(R.id.bnAdd);
        kindList = (ListView) rootView.findViewById(R.id.kindList);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        // 为“添加”按钮的单击事件绑定事件监听器
        bnAdd.setOnClickListener(new OnClickListener()
        {
```

```
            @Override
            public void onClick(View source)
            {
                // 当“添加”按钮被单击时，调用该 Fragment 所在 Activity 的 onItemSelected
                // 方法
                mCallbacks.onItemSelected(ADD_KIND , null);
            }
        });
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "viewKind.jsp";
        try
        {
            // 向指定 URL 发送请求，并把响应包装成 JSONArray 对象
            final JSONArray jsonArray = new JSONArray(
                HttpUtil.getRequest(url));
            // 把 JSONArray 对象包装成 Adapter
            kindList.setAdapter(new KindArrayAdapter(jsonArray
                , getActivity()));
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity()
                , "服务器响应异常，请稍后再试！" ,false);
            e.printStackTrace();
        }
        return rootView;
    }
    // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
    @Override
    public void onAttach(Activity activity)
    {
        super.onAttach(activity);
        // 如果 Activity 没有实现 Callbacks 接口，抛出异常
        if (!(activity instanceof Callbacks))
        {
            throw new IllegalStateException(
                "ManageKindFragment 所在的 Activity 必须实现 Callbacks 接口！");
        }
        // 把该 Activity 当成 Callbacks 对象
        mCallbacks = (Callbacks) activity;
    }
    // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
    @Override
    public void onDetach()
    {
        super.onDetach();
        // 将 mCallbacks 赋为 null
        mCallbacks = null;
    }
}
```

上面程序的 onCreateView()方法中粗体字代码实现了向服务器发送请求，把服务器响应转换成 JSONArray 对象，并使用 ListView 显示物品种类的核心代码。由于物品种类包含的信息量并不大，只有种类名称和种类描述两种，因此程序使用了 KindArrayAdapter 来包装 JSONArray 对象，KindArrayAdapter 提供的列表项既包括种类名，也包括种类描述。下面是该 Adapter 类的代码。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\KindArrayAdapter.java

```
public class KindArrayAdapter extends BaseAdapter
```

```
{
    // 需要包装的 JSONArray
    private JSONArray kindArray;
    private Context ctx;
    public KindArrayAdapter(JSONArray kindArray
        ,Context ctx)
    {
        this.kindArray = kindArray;
        this.ctx = ctx;
    }
    @Override
    public int getCount()
    {
        // 返回 ListView 包含的列表项的数量
        return kindArray.length();
    }
    @Override
    public Object getItem(int position)
    {
        // 获取指定列表项所包装的 JSONObject
        return kindArray.optJSONObject(position);
    }
    @Override
    public long getItemId(int position)
    {
        try
        {
            return ((JSONObject) getItem(position)).getInt("id");
        }
        catch (JSONException e)
        {
            e.printStackTrace();
        }
        return -1;
    }
    @Override
    public View getView(int position, View convertView,
        ViewGroup parent)
    {
        // 定义一个线性布局管理器
        LinearLayout container = new LinearLayout(ctx);
        // 设置为水平的线性布局管理器
        container.setOrientation(1);
        // 定义一个线性布局管理器
        LinearLayout linear = new LinearLayout(ctx);
        // 设置为水平的线性布局管理器
        linear.setOrientation(0);
        // 创建一个 ImageView
        ImageView iv = new ImageView(ctx);
        iv.setPadding(10, 0, 20, 0);
        iv.setImageResource(R.drawable.item);
        // 将图片添加到 LinearLayout 中
        linear.addView(iv);
        // 创建一个 TextView
        TextView tv = new TextView(ctx);
        try
        {
            // 获取 JSONArray 数组元素的 kindName 属性
            String kindName = ((JSONObject)getItem(position))
                .getString("kindName");
            // 设置 TextView 所显示的内容
```

```
                tv.setText(kindName);
            }
            catch (JSONException e)
            {
                e.printStackTrace();
            }
            tv.setTextSize(20);
            // 将 TextView 添加到 LinearLayout 中
            linear.addView(tv);
            container.addView(linear);
            // 定义一个文本框来显示种类描述
            TextView descView = new TextView(ctx);
            descView.setPadding(30, 0, 0, 0);
            try
            {
                // 获取 JSONArray 数组元素的 kindDesc 属性
                String kindDesc = ((JSONObject)getItem(position))
                    .getString("kindDesc");
                descView.setText(kindDesc);
            }
            catch (JSONException e)
            {
                e.printStackTrace();
            }
            descView.setTextSize(16);
            container.addView(descView);
            return container;
        }
    }
```

从上面的程序中的粗体字代码可以看出，该 Adapter 提供的每个列表项不仅包括种类名称，而且包括种类描述，这样用户查看种类时一目了然。

用户查看物品种类时将可看到如图 19.13 所示的界面。



图 19.13　平板上查看物品种类

为了在手机屏幕上运行该应用，程序提供了一个 ManageKind Activity 来包装、显示该 Fragment，ManageKind 的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ManageKind.java

```
public class ManageKind extends FragmentActivity
    implements Callbacks
{
    @Override
    public Fragment getFragment()
    {
        return new ManageKindFragment();
```

```
        }
        @Override
        public void onItemSelected(Integer id, Bundle bundle)
        {
            // 当用户单击 ManageKindFragment 中的“添加”按钮时，系统启动 AddKind Activity
            Intent i = new Intent(this , AddKind.class);
            startActivity(i);
        }
    }
```

正如上面的粗体字代码所示，ManageKind 仅仅是包装、显示 ManageKindFragment，因此在手机屏幕上运行该程序时，将可以看到如图 19.14 所示界面。



图 19.14 手机上查看物品种类

在如图 19.13、图 19.14 所示界面中有一个“添加种类”按钮，当用户单击该按钮时，Fragment 将会回调它所在 Activity 的 onItemSelected()方法，对于在平板上运行的 Activity，onItemSelected()方法将会使用 AddKindFragment 代替 FrameLayout 中的，而在手机上运行的 Activity，onItemSelected()方法将会启动 AddKind Activity（该 Activity 仅仅是包装、显示 AddKindFragment）。

19.6.3 添加种类的 Servlet

添加物品种类的 Servlet 也是调用业务逻辑组件的方法来完成添加的，该 Servlet 的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddKindServlet.java

```
@WebServlet(urlPatterns="/android/addKind.jsp")
public class AddKindServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
        {
            request.setCharacterEncoding("utf-8");
            // 获取请求参数
            String name = request.getParameter("kindName");
            String desc = request.getParameter("kindDesc");
            // 获取系统业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 调用业务逻辑组件的业务方法添加种类
            int kindId = auctionManager.addKind(name , desc);
            response.setContentType("text/html; charset=GBK");
            // 添加成功
            if (kindId > 0)
            {
                response.getWriter().println("恭喜您，种类添加成功!");
            }
            else
            {
                response.getWriter().println("对不起，种类添加失败!");
            }
        }
}
```

上面 Servlet 调用业务逻辑组件添加物品后，直接返回了添加结果的字符串，因此 Android 客户端只要直接显示该添加结果即可。

19.6.4 添加物品种类

添加物品种类的界面上包括两个输入框，用于接受用户输入种类名称和种类描述，并提供添加、取消两个按钮，该界面布局比较简单，此处不再给出界面布局代码。

添加物品种类的 Fragment 的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\AddKindFragment.java

```
public class AddKindFragment extends Fragment
{
    // 定义界面中两个文本框
    EditText kindName, kindDesc;
    // 定义界面中两个按钮
    Button bnAdd, bnCancel;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.add_kind
            , container , false);
        // 获取界面中两个编辑框
        kindName = (EditText)rootView.findViewById(R.id.kindName);
        kindDesc = (EditText)rootView.findViewById(R.id.kindDesc);
        // 获取界面中的两个按钮
        bnAdd = (Button)rootView.findViewById(R.id.bnAdd);
        bnCancel = (Button)rootView.findViewById(R.id.bnCancel);
        // 为“取消”按钮的单击事件绑定事件监听器
        bnCancel.setOnClickListener(new HomeListener(getActivity()));
        bnAdd.setOnClickListener(new OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                // 输入校验
                if (validate())
                {
                    // 获取用户输入的种类名、种类描述
                    String name = kindName.getText().toString();
                    String desc = kindDesc.getText().toString();
                    try
                    {
                        // 添加物品种类
                        String result =  addKind(name, desc);
                        // 使用对话框来显示添加结果
                        DialogUtil.showDialog(getActivity()
                            , result , true);
                    }
                    catch (Exception e)
                    {
                        DialogUtil.showDialog(getActivity()
                            , "服务器响应异常，请稍后再试！" , false);
                        e.printStackTrace();
                    }
                }
            }
        });
```

```
        return rootView;
    }
    // 对用户输入的种类名称进行校验
    private boolean validate()
    {
        String name = kindName.getText().toString().trim();
        if (name.equals(""))
        {
            DialogUtil.showDialog(getActivity() , "种类名称是必填项！" , false);
            return false;
        }
        return true;
    }
    private String addKind(String name, String desc)
        throws Exception
    {
        // 使用 Map 封装请求参数
        Map<String , String> map = new HashMap<String, String>();
        map.put("kindName" , name);
        map.put("kindDesc" , desc);
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "addKind.jsp";
        // 发送请求
        return HttpUtil.postRequest(url , map);
    }
}
```

上面的 Fragment 先对用户输入的种类名称、种类描述进行输入校验，然后调用 addKind() 方法来添加物品种类，该方法利用 HttpUtil 发送 POST 请求完成添加。添加成功后可以看到系统显示如图 19.15 所示的对话框。



图 19.15 在平板上添加种类

为了兼顾手机屏幕，同样需要定义 Activity 来装载、显示该 Fragment，AddKind Activity 的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\AddKind.java

```
public class AddKind extends FragmentActivity
{
    @Override
    public Fragment getFragment()
    {
        return new AddKindFragment();
    }
}
```

正如上面的粗体字代码所示，AddKind 什么都没干，仅仅是装载、显示 AddKindFragment，

在手机上添加种类，如果添加成功，将看到如图 19.16 所示对话框。



图 19.16　在手机上添加种类

19.7　管理拍卖物品

管理拍卖物品包括浏览自己的拍卖物品、添加拍卖物品两大主要功能。Android 客户端主要充当用户交互的客户端：显示当前用户的拍卖物品；添加拍卖物品时提供输入框供用户输入物品名称、物品描述等必要信息。

19.7.1　查看自己的拍卖物品的 Servlet

查看自己拍卖物品的 Servlet 调用业务逻辑组件的业务逻辑方法来获取自己的拍卖物品，接下来该 Servlet 将会把该物品列表包装成 JSONArray 对象，再转换成 JSON 字符串后输出。

下面是该 Servlet 的代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ViewOwnerItemServlet. java

```
@WebServlet(urlPatterns="/android/viewOwnerItem.jsp")
public class ViewOwnerItemServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
        {
            // 获取 userId
            Integer userId = (Integer)request.getSession(true)
                .getAttribute("userId");
            // 获取业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 获取该用户当前处于拍卖中的所有物品
            List<ItemBean> items = auctionManager.getItemsByOwner(userId);
            // 将查询得到的物品封装成 JSONArray 对象
            JSONArray jsonArr= new JSONArray(items);
            response.setContentType("text/html; charset=GBK");
            response.getWriter().println(jsonArr.toString());
        }
}
```

该 Servlet 中粗体字代码调用了业务逻辑方法来获取当前用户、所有处于拍卖中的物品。直接使用浏览器向该 Servlet 发送请求，将可以看到该 Servlet 输出如图 19.17 所示的 JSON 字符串。



图 19.17　JSON 字符串响应

当服务器返回如图 19.17 所示的 JSON 字符串响应之后，Android 客户端就可以把它转化成 JSONArray 对象了，并从中获取详细的物品信息。

19.7.2 查看自己的拍卖物品

查看拍卖物品的界面使用 ListView 来显示物品列表，该界面布局的代码如下。

程序清单：codes\19\AuctionClient\res\manage_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:gravity="center"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<LinearLayout
      android:orientation="horizontal"
      android:gravity="center"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_margin="@dimen/sub_title_margin">
<LinearLayout
      android:orientation="vertical"
      android:gravity="center"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:paddingLeft="12dp">
<TextView
      android:text="@string/manage_item"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<!-- 添加物品的按钮 -->
<Button
      android:id="@+id/bnAdd"
      android:layout_width="85dp"
      android:layout_height="30dp"
      android:background="@drawable/add_item"/>
</LinearLayout>
<Button
      android:id="@+id/bn_home"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="@dimen/label_font_size"
      android:background="@drawable/home"/>
</LinearLayout>
<!-- 显示物品列表的 ListView -->
<ListView
      android:id="@+id/itemList"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>
</LinearLayout>
```

上面的界面布局中定义了一个 ListView 来显示当前用户的拍卖物品。除此之外，该界面还包含了几个按钮，其中一个按钮用于启动添加物品的用户界面。

管理物品的 Fragment 只要通过 HttpUtil 向服务器发送请求，并把服务器响应转换成 JSONArray 对象，再把 JSONArray 对象包装成 Adapter，再使用 ListView 来显示这些物品即可。管理物品的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ManageItemFragment.java

```
public class ManageItemFragment extends Fragment
{
```

```
public static final int ADD_ITEM = 0x1006;;
Button bnHome, bnAdd;
ListView itemList;
Callbacks mCallbacks;
@Override
public View onCreateView(LayoutInflater inflater
      , ViewGroup container, Bundle savedInstanceState)
{
      View rootView = inflater.inflate(R.layout.manage_item
            , container , false);
      bnHome = (Button) rootView.findViewById(R.id.bn_home);
      bnAdd = (Button) rootView.findViewById(R.id.bnAdd);
      itemList = (ListView) rootView.findViewById(R.id.itemList);
      // 为“返回”按钮的单击事件绑定事件监听器
      bnHome.setOnClickListener(new HomeListener(getActivity()));
      bnAdd.setOnClickListener(new OnClickListener()
      {
            @Override
            public void onClick(View source)
            {

                  mCallbacks.onItemSelected(ADD_ITEM , null);
            }
      });
      // 定义发送请求的 URL
      String url = HttpUtil.BASE_URL + "viewOwnerItem.jsp";
      try
      {
            // 向指定 URL 发送请求
            JSONArray jsonArray = new JSONArray(HttpUtil.getRequest(url));
            // 将服务器响应包装成 Adapter
            JSONArrayAdapter adapter = new JSONArrayAdapter(getActivity()
                  , jsonArray, "name", true);
            itemList.setAdapter(adapter);
      }
      catch (Exception e)
      {
            DialogUtil.showDialog(getActivity()
                  , "服务器响应异常，请稍后再试！", false);
            e.printStackTrace();
      }
      itemList.setOnItemClickListener(new OnItemClickListener()
      {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
                  int position, long id)
            {
                  viewItemInBid(position);  //①
            }
      });
      return rootView;
}
// 当该 Fragment 被添加、显示到 Activity 时，回调该方法
@Override
public void onAttach(Activity activity)
{
      super.onAttach(activity);
      // 如果 Activity 没有实现 Callbacks 接口，抛出异常
      if (!(activity instanceof Callbacks))
      {
            throw new IllegalStateException(
```

```
                    "ManagerItemFragment 所在的 Activity 必须实现 Callbacks 接口!");
            }
            // 把该 Activity 当成 Callbacks 对象
            mCallbacks = (Callbacks) activity;
        }
        // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
        @Override
        public void onDetach()
        {
            super.onDetach();
            // 将 mCallbacks 赋为 null
            mCallbacks = null;
        }
        private void viewItemInBid(int position)
        {
            // 加载 detail_in_bid.xml 界面布局代表的视图
            View detailView = getActivity().getLayoutInflater()
                .inflate(R.layout.detail_in_bid, null);
            // 获取 detail_in_bid.xml 界面中的文本框
            TextView itemName = (TextView) detailView
                .findViewById(R.id.itemName);
            TextView itemKind = (TextView) detailView
                .findViewById(R.id.itemKind);
            TextView maxPrice = (TextView) detailView
                .findViewById(R.id.maxPrice);
            TextView initPrice = (TextView) detailView
                .findViewById(R.id.initPrice);
            TextView endTime = (TextView) detailView
                .findViewById(R.id.endTime);
            TextView itemRemark = (TextView) detailView
                .findViewById(R.id.itemRemark);
            // 获取被单击列表项所包装的 JSONObject
            JSONObject jsonObj = (JSONObject) itemList.getAdapter().getItem(
                position);
            try
            {
                // 通过文本框显示物品详情
                itemName.setText(jsonObj.getString("name"));
                itemKind.setText(jsonObj.getString("kind"));
                maxPrice.setText(jsonObj.getString("maxPrice"));
                itemRemark.setText(jsonObj.getString("desc"));
                initPrice.setText(jsonObj.getString("initPrice"));
                endTime.setText(jsonObj.getString("endTime"));
            }
            catch (JSONException e)
            {
                e.printStackTrace();
            }
            DialogUtil.showDialog(getActivity(), detailView);
        }
    }
```

上面的程序中粗体字代码使用了 HttpUtil 向服务器发送请求，并使用 ListView 来显示服务器返回的物品列表。当用户在平板上进入管理物品的界面时，将会看到如图 19.18 所示的界面。

图 19.18 只是列出拍卖物品的物品名，如果用户需要查看该物品的详情，则可以选择单击代表该物品的列表项，程序将会触发 viewItemInBid(position);方法（如上面程序中①号粗体字代码所示），viewItemInBid(position)方法将会启动一个对话框来显示该物品的详情，该对话框中定义了多个文本框，多个文本框显示了该物品的详情。

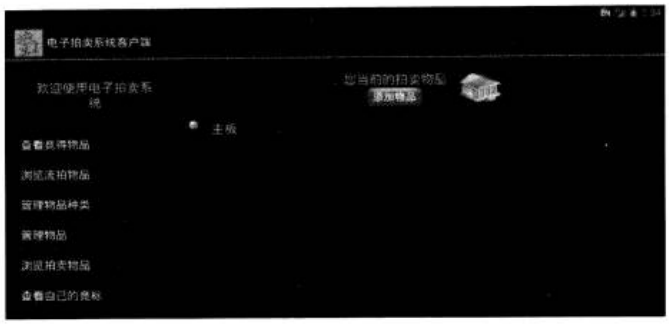


图 19.18　管理自己的拍卖物品

用户单击物品列表的指定物品将可以看到如图 19.19 所示的界面。

为了兼顾手机屏幕，同样需要定义 Activity 来装载、显示该 Fragment，ManageItem Activity 的代码如下。

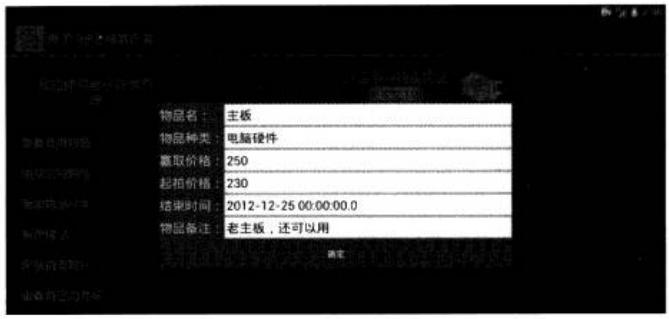


图 19.19　查看物品详情

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ManageItem.java

```
public class ManageItem extends FragmentActivity
      implements Callbacks
{
      @Override
      public Fragment getFragment()
      {
            return new ManageItemFragment();
      }
      @Override
      public void onItemSelected(Integer id, Bundle bundle)
      {
            // 当用户单击“添加”按钮时，将会启动 AddItem Activity
            Intent i = new Intent(this , AddItem.class);
            startActivity(i);
      }
}
```



图 19.20　手机上查看物品详情

在手机屏幕上查看物品功能，将可以看到如图 19.20 所示界面。

图 19.18 所示界面中包含一个“添加物品”按钮，用户单击该按钮即可添加拍卖物品。

19.7.3　添加拍卖物品的 Servlet

添加物品的 Servlet 将会调用业务逻辑组件的方法来添加

物品，添加物品时 Servlet 先解析请求参数，这些请求参数代表了新增物品的属性，再调用业务逻辑方法即可。该 Servlet 的代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddItemServlet.java

```
@WebServlet(urlPatterns="/android/addItem.jsp")
public class AddItemServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
        {
            // 获取 userId
            Integer userId = (Integer)request.getSession(true)
                .getAttribute("userId");
            request.setCharacterEncoding("gbk");
            // 解析请求参数
            String itemName = request.getParameter("itemName");
            String itemDesc = request.getParameter("itemDesc");
            String remark = request.getParameter("itemRemark");
            String initPrice = request.getParameter("initPrice");
            String kindId = request.getParameter("kindId");
            String avail = request.getParameter("availTime");
            // 获取业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 调用业务逻辑组件的方法来添加物品
            int itemId = auctionManager.addItem(itemName ,itemDesc , remark
                , Double.parseDouble(initPrice) , Integer.parseInt(avail)
                , Integer.parseInt(kindId) , userId);
            response.setContentType("text/html; charset=GBK");
            // 添加成功
            if (itemId > 0)
            {
                response.getWriter().println("恭喜您，物品添加成功!");
            }
            else
            {
                response.getWriter().println("对不起，物品添加失败!");
            }
        }
}
```

上面的程序中粗体字代码用于调用业务逻辑方法来添加物品，如果添加成功，该 Servlet 输出登录成功的响应；否则将会输出添加失败的响应。

提供该 Servlet 之后，接下来 Android 客户端就可向该 Servlet 发送请求来添加物品。

19.7.4 添加拍卖物品

添加拍卖物品的程序界面中包含多个文本框，这些文本框用于接受用户输入的物品属性，添加物品的界面布局文件如下。

程序清单：codes\19\AuctionClient\res\layout\add_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
        android:stretchColumns="1">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/add_item_title"
        android:textSize="@dimen/label_font_size"
        android:gravity="center"
        android:padding="@dimen/title_padding"/>
    <!-- 输入物品名称的行 -->
    <TableRow>
    <TextView
        android:text="@string/item_name"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/itemName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>
    </TableRow>
    <!-- 输入物品描述的行 -->
    <TableRow>
    <TextView
        android:text="@string/item_desc"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/itemDesc"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>
    </TableRow>
    <!-- 输入物品备注的行 -->
    <TableRow>
    <TextView
        android:text="@string/remark"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/itemRemark"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"/>
    </TableRow>
    <!-- 输入起拍价格的行 -->
    <TableRow>
    <TextView
        android:text="@string/init_price"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText
        android:id="@+id/initPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"/>
    </TableRow>
    <!-- 选择有效时间的行 -->
```

```
<TableRow>
<TextView
      android:text="@string/avail_time"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<Spinner
      android:id="@+id/availTime"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:entries="@array/availTime"/>
</TableRow>
<!-- 选择选择物品种类的行 -->
<TableRow>
<TextView
      android:text="@string/item_kind"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<Spinner
      android:id="@+id/itemKind"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"/>
</TableRow>
<!-- 定义按钮的行 -->
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center">
<Button
      android:id="@+id/bnAdd"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/add"/>
<Button
      android:id="@+id/bnCancel"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cancel"/>
</LinearLayout>
</TableLayout>
```

上面的界面布局中包含了两个 Spinner 组件，它们定义了两个列表框供用户选择有效时间和物品种类，对于选择有效时间的 Spinner 来说，程序可以直接指定一个数组作为它的列表项；但对于选择物品种类的 Spinner 来说，程序必须加载系统中所有物品种类来作为列表项，因此为了在 Spinner 中加载物品选项，程序必须向/android/viewKind.jsp 发送请求，然后把响应包装成 Adapter，接下来使用 Spinner 来显示种类列表。

当用户在平板上进入添加图书的界面之后会看到如图 19.21 所示的输入界面。

从图 19.21 所示界面可以看出，程序最下面的物品种类就是从系统中加载出来的，当用户在图 19.21 所示的界面中填写了拍卖物品的详情之后，程序将会向/android/addItem.jsp 发送请求来添加物品，添加物品的 Fragment 的代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\AddItemFragment.java

```
public class AddItemFragment extends Fragment
{
    // 定义界面中文本框
    EditText itemName, itemDesc,itemRemark,initPrice;
```

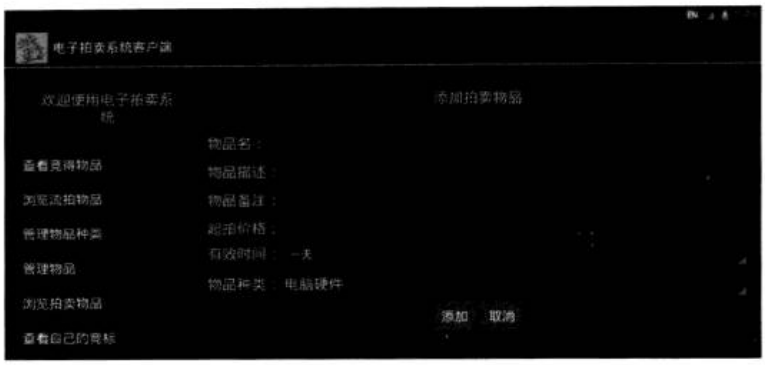


图 19.21　添加拍卖物品

```
Spinner itemKind , availTime;
// 定义界面中两个按钮
Button bnAdd, bnCancel;
@Override
public View onCreateView(LayoutInflater inflater
      , ViewGroup container, Bundle savedInstanceState)
{
      View rootView = inflater.inflate(R.layout.add_item
            , container , false);
      // 获取界面中的文本框
      itemName = (EditText) rootView.findViewById(R.id.itemName);
      itemDesc = (EditText) rootView.findViewById(R.id.itemDesc);
      itemRemark = (EditText) rootView.findViewById(R.id.itemRemark);
      initPrice = (EditText) rootView.findViewById(R.id.initPrice);
      itemKind = (Spinner) rootView.findViewById(R.id.itemKind);
      availTime = (Spinner) rootView.findViewById(R.id.availTime);
      // 定义发送请求的地址
      String url = HttpUtil.BASE_URL + "viewKind.jsp";
      JSONArray jsonArray = null;
      try
      {
            // 获取系统中所有的物品种类
            // 向执行 URL 发送请求，并把服务器响应包装成 JSONArray
            jsonArray = new JSONArray(HttpUtil.getRequest(url));  //①
      }
      catch (Exception e1)
      {
            e1.printStackTrace();
      }
      // 将 JSONArray 包装成 Adapter
      JSONArrayAdapter adapter = new JSONArrayAdapter(
            getActivity() , jsonArray , "kindName" , false);
      // 显示物品种类列表
      itemKind.setAdapter(adapter);
      // 获取界面中的两个按钮
      bnAdd = (Button) rootView.findViewById(R.id.bnAdd);
      bnCancel = (Button) rootView.findViewById(R.id.bnCancel);
      // 为“取消”按钮的单击事件绑定事件监听器
      bnCancel.setOnClickListener(new HomeListener(getActivity()));
      bnAdd.setOnClickListener(new OnClickListener()
      {
            @Override
            public void onClick(View v)
            {
                  // 执行输入校验
                  if (validate())
```

```
                {
                    // 获取用户输入的物品名、物品描述等信息
                    String name = itemName.getText().toString();
                    String desc = itemDesc.getText().toString();
                    String remark = itemRemark.getText().toString();
                    String price = initPrice.getText().toString();
                    JSONObject kind = (JSONObject) itemKind.getSelectedItem();
                    int avail = availTime.getSelectedItemPosition();
                    //根据用户选择有效时间选项，指定实际的有效时间
                    switch(avail)
                    {
                        case 5 :
                            avail = 7;
                            break;
                        case 6 :
                            avail = 30;
                            break;
                        default :
                            avail += 1;
                            break;
                    }
                    try
                    {
                        // 添加物品
                        String result = addItem(name, desc
                            , remark , price , kind.getInt("id") , avail);
                        // 显示对话框
                        DialogUtil.showDialog(getActivity()
                            , result , true);
                    }
                    catch (Exception e)
                    {
                        DialogUtil.showDialog(getActivity()
                            , "服务器响应异常，请稍后再试！" , false);
                        e.printStackTrace();
                    }
                }
            }
        });
        return rootView;
    }
    // 对用户输入的物品名、起拍价格进行校验
    private boolean validate()
    {
        String name = itemName.getText().toString().trim();
        if (name.equals(""))
        {
            DialogUtil.showDialog(getActivity() , "物品名称是必填项！" , false);
            return false;
        }
        String price = initPrice.getText().toString().trim();
        if (price.equals(""))
        {
            DialogUtil.showDialog(getActivity() , "起拍价格是必填项！" , false);
            return false;
        }
        try
        {
            // 尝试把起拍价格转换为浮点数
            Double.parseDouble(price);
        }
```

```
            catch(NumberFormatException e)
            {
                DialogUtil.showDialog(getActivity() , "起拍价格必须是数值！" , false);
                return false;
            }
            return true;
        }
        private String addItem(String name, String desc
            , String remark , String initPrice , int kindId , int availTime)
            throws Exception
        {
            // 使用Map封装请求参数
            Map<String , String> map = new HashMap<String, String>();
            map.put("itemName" , name);
            map.put("itemDesc" , desc);
            map.put("itemRemark" , remark);
            map.put("initPrice" , initPrice);
            map.put("kindId" , kindId + "");
            map.put("availTime" , availTime + "");
            // 定义发送请求的URL
            String url = HttpUtil.BASE_URL + "addItem.jsp";
            // 发送请求
            return HttpUtil.postRequest(url , map);
        }
    }
```

上面的程序中①号粗体字代码向/android/viewKind.jsp发送请求，并把服务器响应包装成JSONArray对象，然后使用Spinner把这些物品种类显示出来即可。

当填写了拍卖物品的详情之后，用户单击“添加”按钮将先执行输入校验，然后调用addItem()方法来添加物品，如上面程序中两行粗体字代码所示。

如果用户添加拍卖物品成功，程序将会看到如图19.22所示的对话框。

为了兼顾手机屏幕，同样需要定义Activity来装载、显示该Fragment，AddItem Activity的代码如下。

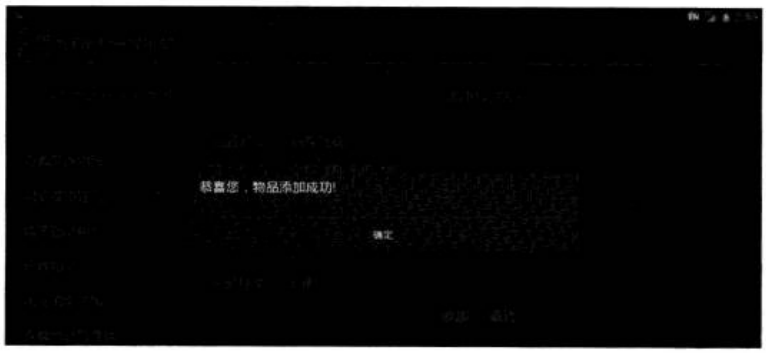


图19.22　添加物品成功

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\AddItem.java

```
public class AddItem extends FragmentActivity
{
    @Override
    public Fragment getFragment()
    {
        return new AddItemFragment();
    }
}
```

从上面的粗体字代码不难看出，AddItem Activity 仅仅是装载并显示 AddItemFragment。在手机上添加物品成功后可以看到如图 19.23 所示界面。



图 19.23 在手机上添加物品成功

19.8 参与竞拍

用户可以通过物品种类来浏览系统中的拍卖物品，找到合适的拍卖物品之后，用户可以对该物品进行竞价。

19.8.1 选择物品种类

前面介绍的/android/viewKind.jsp 可以生成所有物品种类的响应，Android 客户端只要向该 Servlet 发送请求即可显示物品种类供用户选择。

选择物品种类的界面上主要包含一个 ListView，该 ListView 列出系统中全部物品种类，当用户单击某个物品种类时，程序将会显示该种类下的全部物品。

选择物品种类的 Fragment 代码如下。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ChooseKindFragment.java

```
public class ChooseKindFragment extends Fragment
{
    public static final int CHOOSE_ITEM = 0x1008;
    Callbacks mCallbacks;
    Button bnHome;
    ListView kindList;
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.choose_kind
            , container , false);
        bnHome = (Button) rootView.findViewById(R.id.bn_home);
        kindList = (ListView) rootView.findViewById(R.id.kindList);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "viewKind.jsp";
        try
        {
            // 向指定 URL 发送请求，并将服务器响应包装成 JSONArray 对象
            JSONArray jsonArray = new JSONArray(
                HttpUtil.getRequest(url)); //①
            // 使用 ListView 显示所有物品准种类
            kindList.setAdapter(new KindArrayAdapter(jsonArray
                , getActivity()));
        }
        catch (Exception e)
        {
            DialogUtil.showDialog(getActivity()
                , "服务器响应异常，请稍后再试！" , false);
            e.printStackTrace();
        }
        kindList.setOnItemClickListener(new OnItemClickListener()
        {
            @Override
            public void onItemClick(AdapterView<?> parent, View view,
```

```
                    int position, long id)
                {
                    Bundle bundle = new Bundle();
                    bundle.putLong("kindId", id);
                    mCallbacks.onItemSelected(CHOOSE_ITEM , bundle);
                }
            });
            return rootView;
        }
        // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
        @Override
        public void onAttach(Activity activity)
        {
            super.onAttach(activity);
            // 如果 Activity 没有实现 Callbacks 接口，抛出异常
            if (!(activity instanceof Callbacks))
            {
                throw new IllegalStateException(
                    "ManageKindFragment 所在的 Activity 必须实现 Callbacks 接口!");
            }
            // 把该 Activity 当成 Callbacks 对象
            mCallbacks = (Callbacks) activity;
        }
        // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
        @Override
        public void onDetach()
        {
            super.onDetach();
            // 将 mCallbacks 赋为 null
            mCallbacks = null;
        }
}
```

程序中①号粗体字代码用于向指定 URL 发送请求，并将服务器响应转换成 JSONArray 对象，接下来程序只要把 JSONArray 对象包装成 Adapter，并使用 ListView 显示种类列表即可。

当用户单击指定物品种类时，程序将会回调该 Fragment 所在 Activity 的 onItemSelecte() 方法，并把当前物品种类作为参数传过去，而该 Fragment 所在 Activity 将会负责加载 Fragment、或启动对应的 Activity 来显示该种类下的全部拍卖物品。

➤➤19.8.2 根据种类浏览物品的 Servlet

根据种类浏览拍卖物品的 Servlet 调用业务逻辑方法来获取所有物品。该 Servlet 所做的就是调用业务逻辑方法来查询指定种类下的全部物品，并把这些物品包装成 JSONArray，再转换成 JSON 字符串输出。

下面是该 Servlet 的代码。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\ItemListServlet.java

```
@WebServlet(urlPatterns="/android/itemList.jsp")
public class ItemListServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
    {
            request.setCharacterEncoding("gbk");
            // 获取物品种类 ID
```

```
            String kindId = request.getParameter("kindId");
            // 获取业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 调用业务逻辑方法来获取全部物品
            List<ItemBean> items = auctionManager
                .getItemsByKind(Integer.parseInt(kindId));
            // 将物品列表包装成 JSONArray
            JSONArray jsonArr= new JSONArray(items);
            response.setContentType("text/html; charset=GBK");
            response.getWriter().println(jsonArr.toString());
        }
    }
```

上面的程序中粗体字代码调用了业务逻辑方法来获取指定种类的全部物品，接下来程序将这些物品包装成 JSONArray，再转换成 JSON 字符串输出。

直接使用浏览器访问该 Servlet 将看到如图 19.24 所示的 JSON 字符串。

19.8.3 根据种类浏览物品



图 19.24 服务器响应的 JSON 字符串

服务器端 Servlet 生成图 19.24 所示的 JSON 字符串响应后，Android 客户端只要向该 Servlet 发送请求并把该响应转换为 JSONArray 对象，再将该 JSONArray 对象包装成 Adapter，并使用 ListView 来显示这些物品即可。

下面是 ChooseItemFragment 的代码。

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\ChooseItem.java

```
public class ChooseItemFragment extends Fragment
{
    public static final int ADD_BID = 0x1009;
    Button bnHome;
    ListView succList;
    TextView viewTitle;
    Callbacks mCallbacks;
    // 重写该方法，该方法返回的 View 将作为 Fragment 显示的组件
    @Override
    public View onCreateView(LayoutInflater inflater
        , ViewGroup container, Bundle savedInstanceState)
    {
        View rootView = inflater.inflate(R.layout.view_item
            , container , false);
        // 获取界面中的“返回”按钮
        bnHome = (Button)rootView.findViewById(R.id.bn_home);
        succList = (ListView)rootView.findViewById(R.id.succList);
        viewTitle = (TextView)rootView.findViewById(R.id.view_titile);
        // 为“返回”按钮的单击事件绑定事件监听器
        bnHome.setOnClickListener(new HomeListener(getActivity()));
        long kindId = getArguments().getLong("kindId");
        // 定义发送请求的 URL
        String url = HttpUtil.BASE_URL + "itemList.jsp?kindId=" + kindId;
        try
        {
            // 根据种类 ID 获取该种类对应的所有物品
            JSONArray jsonArray = new JSONArray(HttpUtil.getRequest(url));
            JSONArrayAdapter adapter = new JSONArrayAdapter(
                getActivity() , jsonArray , "name" , true);
            // 使用 ListView 显示当前种类的所有物品
```

```
                succList.setAdapter(adapter);
            }
            catch (Exception e1)
            {
                DialogUtil.showDialog(getActivity() , "服务器响应异常，请稍后再试！" , false);
                e1.printStackTrace();
            }
            // 修改标题
            viewTitle.setText(R.string.item_list);
            succList.setOnItemClickListener(new OnItemClickListener()
            {
                @Override
                public void onItemClick(AdapterView<?> parent, View view,
                    int position, long id)
                {
                    JSONObject jsonObj = (JSONObject) succList
                        .getAdapter().getItem(position);
                    Bundle bundle = new Bundle();
                    try
                    {
                        bundle.putInt("itemId" , jsonObj.getInt("id"));
                    }
                    catch (JSONException e)
                    {
                        e.printStackTrace();
                    }
                    mCallbacks.onItemSelected(ADD_BID, bundle);
                }
            });
            return rootView;
        }
        // 当该 Fragment 被添加、显示到 Activity 时，回调该方法
        @Override
        public void onAttach(Activity activity)
        {
            super.onAttach(activity);
            // 如果 Activity 没有实现 Callbacks 接口，抛出异常
            if (!(activity instanceof Callbacks))
            {
                throw new IllegalStateException(
                    "ManagerItemFragment 所在的 Activity 必须实现 Callbacks 接口！");
            }
            // 把该 Activity 当成 Callbacks 对象
            mCallbacks = (Callbacks) activity;
        }
        // 当该 Fragment 从它所属的 Activity 中被删除时回调该方法
        @Override
        public void onDetach()
        {
            super.onDetach();
            // 将 mCallbacks 赋为 null
            mCallbacks = null;
        }
    }
```

上面程序中的粗体字代码用于向指定 Servlet 发送请求，并把服务器响应转换成 JSONArray 对象，再将该对象包装成 Adapter，并使用 ListView 来显示这些物品。

当用户在平板上选择指定物品种类时将会看到如图 19.25 所示的物品列表。



图 19.25 在平板上查看指定种类的拍卖物品

图 19.25 列出了指定种类下所有的拍卖物品，用户可以单击指定物品进入拍卖界面。



提示：

系统也为兼容手机屏幕提供了 ChooseItem Activity，该 Activity 的作用也只是包装并显示 ChooseItemFragment，此处不再给出 ChooseItem 的代码。

19.8.4 参与竞价的 Servlet

参与竞价的 Servlet 调用业务逻辑组件的业务方法来添加竞价记录，如果竞价成功，程序直接生成竞价成功的响应提示；如果竞价失败，程序生成竞价失败的响应提示。参与竞价的 Servlet 代码如下。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\AddBidServlet.java

```
@WebServlet(urlPatterns="/android/addBid.jsp")
public class AddBidServlet extends BaseServlet
{
    public void service(HttpServletRequest request ,
            HttpServletResponse response)
            throws IOException , ServletException
        {
            // 获取 userId
            Integer userId = (Integer)request.getSession(true)
                .getAttribute("userId");
            request.setCharacterEncoding("gbk");
            // 获取请求参数
            String itemId = request.getParameter("itemId");
            String bidPrice = request.getParameter("bidPrice");
            // 获取业务逻辑组件
            AuctionManager auctionManager = (AuctionManager)getCtx()
                .getBean("mgr");
            // 调用业务方法来添加竞价
            int bidId = auctionManager.addBid(Integer.parseInt(itemId)
                , Double.parseDouble(bidPrice) , userId);
            response.setContentType("text/html; charset=GBK");
            // 竞价成功
            if (bidId > 0)
            {
                response.getWriter().println("恭喜您，竞价成功!");
            }
            else
            {
                response.getWriter().println("对不起，竞价失败!");
            }
```

```
        }
    }
```

上面的 Servlet 中粗体字代码调用了业务逻辑方法来添加竞价，每次用户参与竞拍就是添加一条竞价记录，也就是向该 Servlet 发送一个请求。

19.8.5 参与竞价

当用户选择指定物品参与竞价时，系统将会显示该物品的当前详情，例如起拍价、当前最高竞价等，界面的最下方提供一个输入框供用户输入竞拍价。

用于竞价的界面布局文件如下。

程序清单：codes\19\AuctionClient\res\layout\add_bid.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
<TableLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:stretchColumns="1">
<TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/item_detail_title"
      android:textSize="@dimen/label_font_size"
      android:gravity="center"
      android:padding="@dimen/title_padding"/>
<!-- 显示物品名称的行 -->
<TableRow>
<TextView
      android:text="@string/item_name"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<TextView
      android:id="@+id/itemName"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      style="@style/tv_show"/>
</TableRow>
<!-- 显示物品描述的行 -->
<TableRow>
<TextView
      android:text="@string/item_desc"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<TextView
      android:id="@+id/itemDesc"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      style="@style/tv_show"/>
</TableRow>
<!-- 显示物品备注的行 -->
<TableRow>
<TextView
      android:text="@string/remark"
```

```
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/itemRemark"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示物品种类的行 -->
    <TableRow>
    <TextView
        android:text="@string/item_kind"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/itemKind"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示起拍价格的行 -->
    <TableRow>
    <TextView
        android:text="@string/init_price"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/initPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示最高价格的行 -->
    <TableRow>
    <TextView
        android:text="@string/max_price"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/maxPrice"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
    <!-- 显示结束时间的行 -->
    <TableRow>
    <TextView
        android:text="@string/end_time"
        android:textSize="@dimen/label_font_size"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/endTime"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/tv_show"/>
    </TableRow>
```

```
<!-- 输入竞拍价格的行 -->
<TableRow>
<TextView
      android:text="@string/you_bid"
      android:textSize="@dimen/label_font_size"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
<EditText
      android:id="@+id/bidPrice"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:inputType="numberDecimal"/>
</TableRow>
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center">
<Button
      android:id="@+id/bnAdd"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/bid"/>
<Button
      android:id="@+id/bnCancel"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/cancel"/>
</LinearLayout>
</TableLayout>
</ScrollView>
```

从上面的界面布局可以看出，当用户选择竞拍物品之后，程序会显示该物品的详情，界面下方包含一个文本框供用户填写竞拍价格。该界面如图 19.26 所示。

当用户在图 19.26 所示界面中输入竞拍价格，单击“竞价”按钮后，程序将会向 /android/addBid.jsp 发送请求、添加竞价记录。

参与竞价的 Fragment 的代码如下。



图 19.26　参与竞价

程序清单：codes\19\AuctionClient\src\org\crazyit\auction\client\AddBidFragment.java

```
public class AddBidFragment extends Fragment
{
      // 定义界面中文本框
      TextView itemName, itemDesc,itemRemark,itemKind
            ,initPrice , maxPrice ,endTime;
      EditText bidPrice;
      // 定义界面中两个按钮
      Button bnAdd, bnCancel;
```

```
// 定义当前正在拍卖的物品
JSONObject jsonObj;
@Override
public View onCreateView(LayoutInflater inflater
      , ViewGroup container, Bundle savedInstanceState)
{
      View rootView = inflater.inflate(R.layout.add_bid
            , container , false);
      // 获取界面中编辑框
      itemName = (TextView)rootView.findViewById(R.id.itemName);
      itemDesc = (TextView)rootView.findViewById(R.id.itemDesc);
      itemRemark = (TextView)rootView.findViewById(R.id.itemRemark);
      itemKind = (TextView)rootView.findViewById(R.id.itemKind);
      initPrice = (TextView)rootView.findViewById(R.id.initPrice);
      maxPrice = (TextView)rootView.findViewById(R.id.maxPrice);
      endTime = (TextView)rootView.findViewById(R.id.endTime);
      bidPrice = (EditText)rootView.findViewById(R.id.bidPrice);
      // 获取界面中的两个按钮
      bnAdd = (Button)rootView.findViewById(R.id.bnAdd);
      bnCancel = (Button)rootView.findViewById(R.id.bnCancel);
      // 为“取消”按钮的单击事件绑定事件监听器
      bnCancel.setOnClickListener(new HomeListener(getActivity()));
      // 定义发送请求的 URL
      String url = HttpUtil.BASE_URL + "getItem.jsp?itemId="
            + getArguments().getInt("itemId");
      try
      {
            // 获取指定的拍卖物品
            jsonObj = new JSONObject(HttpUtil.getRequest(url));
            // 使用文本框来显示拍卖物品的详情
            itemName.setText(jsonObj.getString("name"));
            itemDesc.setText(jsonObj.getString("desc"));
            itemRemark.setText(jsonObj.getString("remark"));
            itemKind.setText(jsonObj.getString("kind"));
            initPrice.setText(jsonObj.getString("initPrice"));
            maxPrice.setText(jsonObj.getString("maxPrice"));
            endTime.setText(jsonObj.getString("endTime"));
      }
      catch (Exception e1)
      {
            DialogUtil.showDialog(getActivity(), "服务器响应出现异常!", false);
            e1.printStackTrace();
      }
      bnAdd.setOnClickListener(new OnClickListener()
      {
            @Override
            public void onClick(View v)
            {
                  try
                  {
                        // 执行类型转换
                        double curPrice = Double.parseDouble(
                              bidPrice.getText().toString());
                        // 执行输入校验
                        if( curPrice < jsonObj.getDouble("maxPrice"))  //①
                        {
                              DialogUtil.showDialog(getActivity(),
                                    "您输入的竞价必须高于当前竞价", false);
                        }
                        else
                        {
```

```
                                    // 添加竞价
                                    String result = addBid(jsonObj.getString("id")
                                          , curPrice + ""); //②
                                    // 显示对话框
                                    DialogUtil.showDialog(getActivity()
                                          , result , true);
                              }
                        }
                        catch(NumberFormatException ne)
                        {
                              DialogUtil.showDialog(getActivity()
                                    , "您输入的竞价必须是数值", false);
                        }
                        catch(Exception e)
                        {
                              e.printStackTrace();
                              DialogUtil.showDialog(getActivity()
                                    , "服务器响应出现异常，请重试！", false);
                        }
                  }
            });
            return rootView;
      }
      private String addBid(String itemId , String bidPrice)
            throws Exception
      {
            // 使用Map封装请求参数
            Map<String , String> map = new HashMap<String, String>();
            map.put("itemId" , itemId);
            map.put("bidPrice" , bidPrice);
            // 定义请求将会发送到addKind.jsp页面
            String url = HttpUtil.BASE_URL + "addBid.jsp";
            // 发送请求
            return HttpUtil.postRequest(url , map);
      }
}
```

从上面的程序可以看出，当用户单击“竞价”按钮后，程序将先执行输入校验，再调用addBid()方法来添加竞价记录，如上面的程序中①、②两行粗体字代码所示。

如果用户竞价成功，系统显示如图19.27所示的对话框。

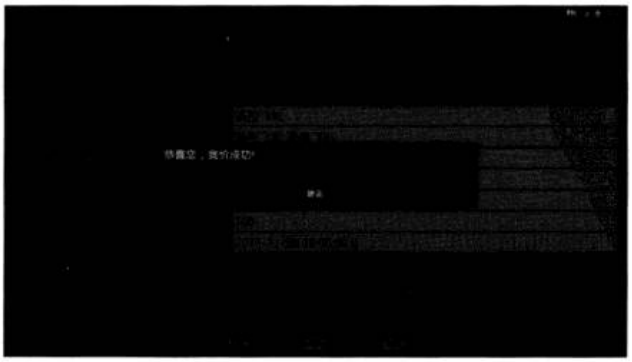


图19.27　竞价成功



提示：

系统也为兼容手机屏幕提供了AddBid Activity，该Activity的作用也只是包装并显示ChooseItemFragment，此处不再给出该AddBid的代码。

19.9 权限控制

前面介绍时已经看到：服务器端程序在处理用户登录时，如果用户登录成功，系统会把用户 ID 放入 HTTP session 中，方便系统跟踪用户的登录状态。

对于 Android 客户端程序来说，由于 Android 客户端采用了 Apache HttpClient 来发送请求、获取响应，因此 HttpClient 会自动维护与服务器之间的登录状态。在有效时间之内，服务器端程序可以跟踪到 Android 客户端的登录状态。

本系统要求只有登录用户才能使用系统功能，因此程序考虑在服务器端使用 Filter 进行控制，Filter 只要拦截匹配的/android/*（匹配该 URL 的 Servlet 向 Android 客户端提供响应）的 URL 即可。

程序清单：codes\19\auction\WEB-INF\src\org\crazyit\auction\servlet\Authority.java

```
@WebFilter(urlPatterns="/android/*")
public class Authority implements Filter
{
    public void init(FilterConfig config)
        throws ServletException
    {
    }
    public void doFilter(ServletRequest request,
        ServletResponse response, FilterChain chain)
        throws IOException , ServletException
    {
        HttpServletRequest hrequest = (HttpServletRequest)request;
        // 获取 HttpSession 对象
        HttpSession session = hrequest.getSession(true);
        Integer userId = (Integer)session.getAttribute("userId");
        // 如果用户已经登录，或用户正在登录
        if ((userId != null && userId > 0)
            || hrequest.getRequestURI().endsWith("/login.jsp"))
        {
            // “放行”请求
            chain.doFilter(request , response);
        }
        else
        {
            response.setContentType("text/html; charset=GBK");
            // 生成错误提示。
            response.getWriter().println("您还没有登录系统，请先系统！");
        }
    }
    public void destroy()
    {
    }
}
```

正如上面的粗体字代码看到的，Filter 要求 HTTP session 中的 userId 属性不为 null，且 userId 属性大于 0，这样就可以判定该用户已经登录系统，否则该 Filter 不会“放行”请求，而是直接向客户端生成提示信息。

注意：

笔者见过一些与该应用类似的 Android 应用的权限控制，它们并没有在服务器端进行权限控制，而是只要求用户在第一次使用时登录系统，但实际上这

是不够的。如果应用程序不在服务器端进行权限控制，而是只在客户端要求用户登录，那么这种系统的安全控制十分脆弱。

对于绝大部分普通用户来说，这样的安全控制可能没有太大的问题；但对于恶意用户来说，他可以采用多种方法来绕过客户端的登录要求——最简单的方法比如反编译 Android 应用，让用户在启动程序时立即进入 Main Activity，如果服务器不再进行权限控制，那么整个应用就“赤裸裸”地暴露出来了，这将是一件可怕的事情。所以这里系统不仅要求用户第一次使用时进行登录，而且程序还在服务器端进行了控制，这样才可保证系统的安全性。



19.10　本章小结

本章介绍了一个非常实用的 Android 应用，Android 应用充当电子拍卖系统的客户端，服务器端则采用 Struts 2 + Spring + Hibernate 的技术组合，架构上采用了控制器层、业务逻辑层、DAO 层的分层架构，保证整个项目具有极好的可扩展性和可维护性。由于本书是一本介绍 Android 开发的图书，因此本章并未介绍服务器端的业务逻辑组件、DAO 组件的实现，而是重点介绍了 Android 客户端的实现，包括为 Android 客户端提供响应的 Servlet 实现，Android 客户端的界面布局，Activity 实现等。本章的 Android 客户端通过 Apache HttpClient 与服务器交互，服务器与客户端之间采用 JSON 作为数据交换格式。读者学习本章需要重点掌握 Android 应用与传统企业应用整合的方式，二者之间通过网络进行数据交换的方式。

